



Development Standards Manual

Version 7.5

© 2008 Open Systems Holdings Corp. All rights reserved.

Document Number 2810.DS75

No part of this manual may be reproduced by any means without the written permission of Open Systems, Inc.

OPEN SYSTEMS and OSAS are registered trademarks and Resource Manager, Resource Manager for Windows, and Report Writer are trademarks of Open Systems Holdings Corp. All other marks are the property of their respective owners.

Printed in U.S.A. This manual is recyclable.

September 2008, Release 7.5

This document has been prepared to conform to the current release version of OPEN SYSTEMS Accounting Software. Because of our extensive development efforts and our desire to further improve and enhance the software, inconsistencies may exist between the software and the documentation in some instances. Call your customer support representative if you encounter an inconsistency.

Open Systems, Inc.
4301 Dean Lakes Blvd.
Shakopee, Minnesota 55379

General Telephone	(952) 403-5700
General Fax	(952) 496-2495
Support Telephone	(800) 582-5000
Support Fax	(952) 403-5870
Internet Address	www.osas.com

CONTENTS

Chapter 1	Introduction	1-1
Chapter 2	OSAS Standards	2-1
	Program Structure	2-11
	Coding Conventions	2-39
	Environment	2-71
	Data Files	2-91
	Installation	2-101
	Using String Templates	2-119
	Data File Conversion	2-125
	Report and Screen Conventions	2-133
Chapter 3	Routines	3-1
	General Routines	3-3
	Application-Specific Routines	3-129
Chapter 4	Upgrading OSAS	4-1
	OSAS 4.5x Upgrade Procedure	4-3
	Converting to 6.xx Graphical	4-19
	Converting to 7.0x	4-59
	Converting to 7.5x	4-65
Chapter 5	Technical Documentation	5-1
	Sample Specification	5-9
Chapter 6	Documentation	6-1
Chapter 7	Quality Assurance	7-1
	Testing Phase	7-3
	Design Phase	7-5
	Acceptance Testing	7-13
	QA Terminology	7-17

CHAPTER 1

1

OSAS Development Standards	1-1
Operating Systems	1-2
Code Listings	1-2

Introduction

OSAS Development Standards

The following chapters describe the coding standards that Open Systems, Inc. uses for the OPEN SYSTEMS[®] Accounting Software (OSAS[®]) line of products written in the Business Basic Extended language.

Language

OPEN SYSTEMS Accounting Software is written in Business Basic, an efficient and powerful tool for building business applications.

Our current language vendor is BASIS International, Ltd. of Albuquerque, New Mexico, whose current interpreters are called Business Basic Extended (BB^x) Visual PRO/5[®] (for Windows), PRO/5[®] (for other operating systems), and Business Basic Java (BBj) (for all operating systems). BASIS provides a powerful, high-quality version of the Business Basic language.

Memory Allocation Scheme

The memory configuration for OSAS version 7.5 under Pro5 and Visual Pro5 is 1 MB of memory for the combined data and program workspace, which is invoked by using the **-m4096** parameter on the BB^x command line.

The **-m** parameter controls the main workspace. If used alone, it contains both programs and data. If used in conjunction with the **-p** parameter, it contains only data; a separate workspace is created for programs based on the **-p** parameter. The number following the **-m** or **-p** parameters represents the number of memory pages allocated. For example, **-m4096** provides pages 0-4095, for a total of 4096 pages; each page is 256 bytes providing for a total memory of 1 MB.

The balance of the available memory is called *system memory*. It is allocated as it is needed by all the software running on the system. BB^x uses space here for many internal purposes. Windows and public programs use space here too.

OSAS task structure, window usage, and public program design run best with a 1 MB main workspace.

The use of memory in the main workspace is dynamic. If a program requiring additional memory is loaded, the interpreter uses all the memory up to the current size of the data. The data area can grow until it reaches the area containing the current program. If the data and program workspaces are separate, each can grow independently to the size of its respective workspace.

Under BBj, no memory configuration is needed, as BBj thru Java handles all memory management.

Operating Systems

OSAS is designed to run on the most useful and popular operating systems—currently Windows 2000/XP/Vista, Novell NetWare, and UNIX/AIX/Linux.

BB^x handles most of the operating system tasks, providing a stable, powerful, and portable platform for development.

Read the BB^x PRO/5 and BBj reference manuals to find out how to define and properly use the operating system and the various devices.

Code Listings

Code examples in the following chapters appear in a different typeface than the surrounding text.

When a line of code is continued on the next line, the continuation line shows the line number and a colon (:). Use the following guidelines when typing in lines of code from this document:

- For normal lines of code, type all the characters on the line and then press **Enter**.
- For lines continuing from one line to the next, type all the characters on the line—do not press **Enter**. Then continue with the first character after the colon on the continuation line. Ignore the repeated line numbers and trailing colons. Press **Enter** after the last character of the total line.

Spaces are significant in program samples.

This is an example of a source code listing:

```
0010 REM O "RMACCOMP      - 04/04/89 - Add/Change Company Information"
0899 REM O - (RMFIL) 899-938
0900 RMFIL: REM  03/29/89 - Resource Manager file iolists
0902 OSAPPL: IOLIST  APPLCODE$,APPLDESC$,APPLFILE$,APPLFLAG$,APPLVER$
0904 OSCODE: IOLIST  CODECOMP$,CODECODE$,CODEMENU$,CODENAME$,CODEPARM$
0906 OSCOMP: IOLIST  COMPID$,COMPNAME$,COMPADDR1$,COMPADDR2$,COMPCITY$,
0906:COMPSTATE$,COMPZIP$,COMPCTRY$,COMPPHONE$,COMPDATE$
0908 OSINFO: IOLIST  INFODEV$,INFOPROG$,INFODATA1$,INFODATA2$,INFODATA3$,INFORW$
```


CHAPTER 2

2

Program Structure	2-11
Coding Conventions	2-39
Environment	2-71
Data Files	2-91
Installation	2-101
Using String Templates	2-119
Data File Conversion	2-125
Report and Screen Conventions	2-133

OSAS Standards

Product Definitions

The first step in creating a product is deciding what it will do and how it will look. Applications can be defined as standalone pieces or as products that integrate with existing OSAS products. This section explains the differences in the product types and the considerations involved in creating a new product.

OSAS products must be

- written in PRO/5, Visual PRO/5, or BBj
- well-designed
- fully debugged
- able to run in both character and GUI modes
- able to meet the industry-specific needs of particular users
- installable using the standard OSAS installation utilities
- consistent in appearance and design with existing OSAS products
- consistent with Open Systems coding, documentation, testing standards and procedures

When designing a new product and creating your initial product specification, consider these questions:

What is the market for this product?

- What is the demand for this product?
- How was this demand determined?

What tasks will this product perform?

- What new functions does it provide?
- What features will it replace?
- What features will it duplicate?

How will this product appear to the user?

- What menus or menu items do we have to create?
- What existing menu items can be used?
- How should the new menus be arranged?

What existing items can be used in this product?

- What OSAS applications must change to work with this new product?
- What existing data files will this product access?
- Which OSAS applications must be installed before this product works?
- Which OSAS applications are enhanced by this product?

What new data storage requirements does this product have?

- How much new data needs to be stored?
- What reserved fields are available in the existing data files?

- What new files are required?
- Which applications can use these new files?

What new forms does this product require?

- Can the product be adapted to use existing preprinted forms?
- Which reports are enhanced or replaced by this product?

How will this product fit into the OSAS product line?

- Does it duplicate an existing product?
- Does it render an existing product unusable?
- Is it too large (or small) a task for one application?
- With what other application should it interface?

Application Types

Once you have chosen the initial concept for your product, read the standard product definitions below to decide which type of product you are creating.

Standard Application

An application is a logical collection of accounting, business operation, or information processing tasks, usually defined by generally accepted accounting principles or standard business needs and practices.

A standard application must have its own main menu, programs, BRC files, data files, sample data files, **xxTB** table file, GENERAL Report Writer dictionary files and ODBC dictionary files.

In addition, a standard application usually contains one or more of these items:

- a master file

- a transaction file
- a history file
- miscellaneous files
- add/change tasks for all files
- file contents list tasks for all files
- enter transaction tasks
- useful reports based on the application's intent
- a post task

It may also include these features:

- interface with other applications--post to or verify/read from
- month-/year-end maintenance
- allow interface of data files with other applications (if the Options switches are set correctly)

A standard application is available from the OSAS main menu. It has a unique two-character application ID used to name the programs and data files.

Add-on Application

An add-on application adds or extends an existing application's functionality. The new application is usually a new submenu or menu added to the main application's menu. Some tasks may be rearranged when the add-on is installed so that tasks are arranged logically where they will be most useful. An example of an add-on application is Sales Order; it is an add-on to Accounts Receivable.

Here is a list of some of the requirements of an add-on application:

- Provide new programs and data files.

- Ship **xxDATA.yyy** (where **yyy** is the version number), or modify **xxDATA.yyy** to include information about new files.
- The user may be required to reset the application options.
- File conversion may be required to convert files to the new layouts.
- Users must be able to add a module to an existing file set easily and create defaults in new files that match existing data.

Enhancement Kit

An enhancement kit consists of enhanced programs, new programs, and menu additions. Here is a list of rules for enhancement kits:

- Install new features or tasks.
- Never overwrite an **xxTB** file. You can merge records into the table file.
- Never change a file IOLIST. You can use existing reserved fields and/or expand existing string fields to handle more data if the record length is greater than with all the fields full.
- Install simply. Copy new programs, add menu items, and provide a utility to convert data files if needed. Do not require the user to convert files or reset options.

Combination Products

A product can be a single application, an add-on application, or an enhancement kit. It can also be a combination of two of these three possibilities. For example, you can design an application that requires enhancements to other existing applications.

Suppose that product **XX** is a new application that provides its own features and data files, and adds new functionality to an existing field in the AR Customer file. If the user uses product **XX** as a stand-alone application, the expanded field usage does not apply. If the user uses Accounts Receivable, there is a conflict between the way product **XX** and AR use the field in the Customer file. To resolve this conflict, you would issue a product **XX/AR** enhancement kit.

Open Systems requires that each application modification remain a part of that application, but you must define it as a separate installable piece. The product package must contain two sets of media: the new application and the enhancement kit that is installed *if* the user also has the application to which the enhancement kit applies.

In operation the user can install the main product just like any other stand-alone application and run it without installing the enhancement. If the user does not have the application to which the enhancement kit applies, he *cannot* install the enhancement kit. If the user does have the application to which the enhancement kit applies or if he adds it at a later date, he *must* install the enhancement kit to get the enhanced functionality.

Never modify Resource Manager to accommodate unusual installations. Add tasks to the **xxFRST.TXT** and **xxLAST.TXT** files instead.

Application ID

Each application is identified by a unique two-letter abbreviation. The descriptive name of an application should be fewer than 40 characters so that it fits on a menu. Here is a list of the current applications and the identifier assigned to each:

Main Applications

Identifier	Application Name
AP	Accounts Payable
AR	Accounts Receivable
BA	Banking
BK	Bill of Materials/Kitting
BR	Bank Reconciliation
CJ	Contractor's Job Cost
DD	Direct Deposit - PA add-on
FA	Fixed Assets
GL	General Ledger
IN	Inventory
JO	Job Cost
LC	Landed Cost
PA	Payroll
PO	Purchase Order - AP add-on
SO	Sales Order - AR add-on
ST	State Tax Routines - PA add-on (prior to v7.0)

Installation, Utilities and Report Generation

Identifier	Application Name
RM	Resource Manager
OW	OSAS Web
GN	GENERAL Report Writer
SD	Software Development Utilities

Version Numbers

The product version number is made up of three parts:

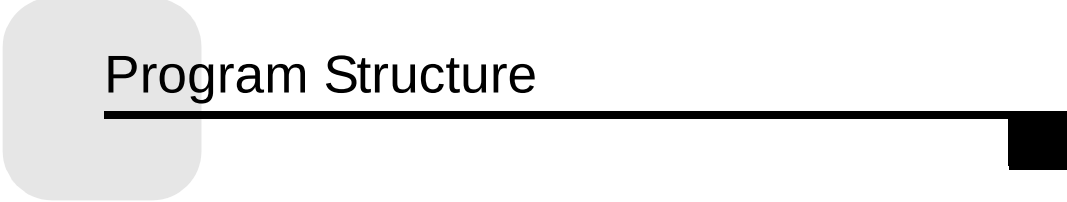
- The integer stands for the design level of the OSAS product. Change it only when significant enhancements are made to the entire set of applications and when programs or data files are incompatible at the interpreter level. These changes cause major compatibility problems with previous levels of the product.
- The tenths decimal place is reserved for system-wide major enhancements. Change it when significant changes cause minor compatibility problems with previous major revisions. Data file conversion programs are provided, but program files must be compatible from 6.(n) to 6.(n+1).
- The hundredths decimal place indicates small changes to the software. Give correction cycle upgrades and minor enhancements a new version number to make sure that the upgraded product is installed and handled as a unique item. All data is compatible if only the minor revision number changes.

The version numbers given to products must be in sync with the OSAS products for two reasons:

- The Resource Manager makes some assumptions about version numbers during installation.
- The functionality of a program depends on the common routines that it uses. The version number also reflects the standards that the product carrying it contains.

Reinstallations and enhancements must be controllable. We attain some control by forcing compatible products to be in particular version number ranges. A product's version numbers must match the version number of the standards to which it adheres.

If a product is an enhancement to an OSAS product, it must use a version number greater than or equal to the version number of the OSAS product it modifies.



Program Structure

Program Names

Most program files in OPEN SYSTEMS Accounting Software are named consistently. The first two letters are the application identifier defined in the previous section. The third and fourth (and sometimes fifth) letters identify a program type. The fifth, sixth and seventh (or sixth, seventh and eighth) characters make the name unique for the task. Each program name chain performs one menu task.

You can also use the sixth character of a program name (n) to differentiate individual items of a task chain. If the program is the first or only program of a program chain, leave off the identifier. Assign the identifier an increasing number or letter if greater than 9. The order in which you normally run the programs should dictate the order in which you number them. You can also use a descriptive identifier of one to three characters to identify programs in a chain.

Here are suggested standard names for program groups:

Character Position

AAACX _n	File Maintenance (Add/Change) program
AADeL _n	Delete Transactions
AAENT _n or AAENT _{xxx}	Enter Transactions
AALST _n	List Master File program
AAMAINT _n	Month-/Year-End Maintenance program
AAPRX _n or AAPXX _n	Print program
AAPST _n or AAPST _{xxx}	Post program
AASTM _n	Statement Printing
AAVFY _n or AAVER	Verify program

The letter *x* is a unique character that identifies the master file or the report associated with the current task. For example, in ARACC the fifth character stands for *customer*. In ARPRL the fifth character stands for *labels*.

Make sure that the characters selected for this purpose do not create a conflict with a data file. The data file's fifth, sixth, and seventh characters can include A–Z and 0–9. For example, if APVER is a program that verified new invoices for AP, and APVEx is a data file, company R would not be available to Accounts Payable. In this case use a different character combination, such as APVFY, to denote function. Be careful, too, of long company IDs. We recommend not using the first four letters of a data file when naming programs for an application.

Public programs are named differently from other programs. The standard method to label public programs is to add the extension **.PUB**. Public program names can be eight characters long plus the extension.

Public programs that provide **F6 Maintenance** functionality have the letters *FM* in the sixth and seventh or seventh and eighth characters of the file name. For example, a version of the ARACC program modified to provide **F6 Maintenance** functionality would become ARACCFM.PUB.

General Program Model

The standard model for each program, and section information for each program type, is described below. The line numbers used in all programs are listed below. You use some lines of code for different purposes, depending on the program type. Resource Manager and system routines may differ from this organization

<i>Remarks</i>	0001	SETERR xxxx
	0010–0029	Program identification
	0030–0089	Comments describing program
<i>Initialization</i>	0090–0099	Interprogram control logic and last year file logic
	0100–0199	Open files (see GENOPEN in “Coding Conventions”)
	0200–0299	Variable initialization
	0300–0399	Define functions (DEF FNx)
	0400–0898	(see “Specific Program Models” later in this section)
	0899–0999	IOLISTs
<i>Main Processing</i>	1000–5999	See “Specific Program Models” on page 2-26 for more information
<i>General Routines</i>	6000–6199	GENINPUT routine
	6200–6299	GENINFO routine
	6700–6799	GENLINE routine
<i>Reserved</i>	6300–6399	Reserved
	6400–6499	Reserved
	6500–6699	Additional subroutine area
	6800–7000	Additional subroutine area
	7000–7499	Reserved for future routine

<i>Screen</i>	7500–7699	Print prompts
	7700–7899	Print data
	7900–7999	Print lines
<i>General Routines</i>	8000–8299	Unused
	8300–8399	GENOPEN routine
	8400–8499	GENERROR routine
	8500–8999	See “Specific Program Models” on page 2-26 for more information
	9000–9199	GENDATE routine
	9200–9299	Delete record area
	9300–9399	GENLOCK routine
	9400–9599	GENPC routine
	9600–9989	Reserved for use during program development
<i>Exit</i>	9990–9998	Exit from program (usually RUN O1\$)
	9999	END statement
<i>General Routine</i>	10000–10099	GENNUM routine
	10100–10245	GENPHONE routine
<i>GUI Screen</i>	17500–17699	Print GUI prompts
	17700–17899	Print GUI data
<i>GUI Procedures</i>	18000–18099	Disable GUI fields
	18100–18199	Enable GUI fields
	18200–18250	Clear STBL(“DESTID”)
	18400–18450	Hide or destroy GUI Windows
	18500–18555	Set up GUI_BUTTON\$[ALL] array

GUI Functions	18800-18850	Checks or unchecks a check box
	18850-18890	Checks or unchecks selections in a radio group
	18900-18950	Selects a value from a list button
<i>VAR Usage Area</i>	20000–29999	VAR Modification Area (see below)
<i>Future Space</i>	30000–59999	Additional subroutine area
	60000–64299	Reserved for future utilities
	64300–64399	Used by _EDIT
	64400–65534	Reserved for future use
	65535	Internal END statement (PRO/5 only)

You do not have to include a general routine in a program if it is not used. Treat each general routine as a black box. Merge in the entire routine and use it as is. Do *not* modify a general routine.

Some general routines work with other routines. User input in a program requires GENINPUT and GENERERROR. They work together and must coexist. GENLOCK and GENERERROR usually are not in the same program.

Five-digit line numbers, above line 9999, are used by GENNUM, GENPHONE, and the GUI functions in versions 6.0 and above. We suggest that you use line numbers 20000–29999 for program modifications to avoid conflict with the base programs in the future.

Code Section Details

Fixed groups of code are areas in the main program that are used for specific purposes. Because they provide a consistent method of performing particular tasks, the user can expect a consistent response to a similar condition regardless of the application.

Most code is executed in a linear fashion. The fixed routines are accessed in one of these ways: GOSUB, GOTO, SETERR or ERR=.

Remarks**0010–0029** **Program identification**

The standardized method of identifying each program helps you administratively. Below is a sample of the identification area and a description of each line. The program identification area uses line numbers 0010–0029. Open Systems reserves line numbers 0010–0024.

```
0001 SETERR GENERROR
0010 REM "GLACG - 02/13/2003 - G/L MASTER FILE ADD/CHANGE"
0011 REM "Authors - R1 - TRS, TAB, DSL, AWP, GL, DJB"
0015 REM 0 - (COPYRIGHT) 0015-0029
0016 REM "Standards 7.50"
0019 REM "OPEN SYSTEMS (R) Accounting Software"
0020 REM "Copyright (c) 1982 - 2008 Open Systems Holdings Corp."
0021 REM "All Rights Reserved"
0022 REM "OPEN SYSTEMS is a registered trademark of Open Systems Holdings Corp."
```

0010 **Program name and description**

GLACG	the full program name, including the extension when applicable
02/13/2003	the last date this program was modified

0011 **Author and copyright**

R1	the QA round number of the program
TRS	the original author's initials
TAB, . . . , DJB	the programmers who have made major changes

0015 **Copyright signature line, used by the library manager****0016** **Standards version number**

Standards 7.50	the standards version number

0019 **Trademark line**

Enter exactly as shown.

0020Copyright

1982 - 2003

the original copyright year, followed by a hyphen, followed by the most recent copyright year

0021–0022Copyright and Open Systems trademark line

Enter exactly as shown.

Follow these rules when you change lines in the program identification area:

- When you change a program, change the development date to the date you made the change. The operating system date (the date the program was saved) should match the date on line 10.
- Change the round number when you change a program and submit it to QA.
- Add the author's initials only when you modify the program significantly; which usually corresponds to a major revision.

Copyrights

As a developer creating products for distribution by Open Systems (or with the approval of Open Systems) or for use with OSAS, you may use copyrighted Open Systems programs and modules in your product, *but you must not remove the Open Systems copyright notices, and you must have licensed products.*

All copyrights are and shall remain the exclusive property of Open Systems Holdings Corp. You must acknowledge all copyrights of Open Systems in a form substantially similar to this:

- Product name
- Copyright years
- Open Systems Holdings Corp. All rights reserved.

Any unauthorized use or duplication constitutes infringement of Open Systems copyrights. Open Systems will prosecute anyone responsible for infringing its copyrights. Open Systems conveys authorization to use Open Systems programs by contract (between the ISV and Open Systems) and by licensing the end user.

To avoid infringing Open Systems copyrights, you have two alternatives:

- Enter into a contract with Open Systems authorizing you to use Open Systems copyrighted matter (for example, program code).
- Be sure that the licensee's product contains the section of code shown below. Failure to report unauthorized use may make you liable for copyright infringement.

Add code lines 13 and 14 in OSAS programs that you have modified but not renamed. They must contain the modification information as defined here:

```
0013 REM "[Programmer initials] - [Date last changed] - [your product's name]"
0014 REM "Copyright [year], [your company name]"
```

Any OSAS program you have renamed for other use or used as a template to build a new program must show the old name on line 14 of the new program.

If you modify an Open Systems software program or application and sell it to a user or install it on a system, you must license the original source program and application for use by that user on that system. Failure to do so constitutes an infringement of Open Systems copyrights.

End-User License

These are the conditions of an end-user license:

- The user may operate and modify the software for internal use only.
- You may not post software to a higher-level machine unless you purchase a licensed upgrade for that machine.
- You may not transfer software to another machine unless you remove it from the original machine.
- You may not sell, resell, or barter software unless Open Systems approves a license transfer from one entity to another.

ISV License

An independent software vendor can purchase a license to modify Open Systems software from Open Systems in two ways:

- Contact Open Systems directly for a special-use contract. License rights provided are dependent on the contract.
- Purchase the appropriate version of the Advanced Developer Tool Kit and submit the appropriate information to Open Systems. Once approved by Open Systems, the ISV may modify, reuse, and resell Open Systems code, file structures, and general routines. However, the ISV may do so only in new enhancements, modifications, or applications installed over a licensed copy of the source program, data file, and application used (no exceptions). You must sell and license the base Open Systems application for the user before using software based on OSAS programs on that system.

Any other use of Open Systems software programs, data structures, general routines, user's guides, marketing literature, or other copyrighted product component constitutes copyright infringement. Open Systems will pursue legal action to protect its copyrights.

For your protection, include your own remarks and copyrights at the top of new programs that don't use *any* Open Systems code. The normal remark positions are lines 10–29.

0030–0089 Comments describing program

Use line numbers 0030–0089 for program-specific identification purposes. Use this area to describe the separate pieces of the function, the inter-program control (if any), and the important non-file variables used.

Initialization

0090–0099 Display screen/inter-program control/last-year file logic

The title assignment and the call to draw the screen occur here. The screen title should be assigned to the variable TITLE\$. In version 5.00 and above you can set TITLE\$=OT\$; OT\$ is the function title as passed from the RMMENU program.

When chaining between many programs in the same task, sometimes you must know the level of reentry being performed. The program control variable P9 is used. These lines of code can contain tests on P9 that will control where execution is passed to next.

In General Ledger, Payroll, and other applications the system must check for the use of last- or current-year files. This logic is added here.

0100–0299 Open files (see GENOPEN)

All files should be opened in this section of the program. Use the file opening standards in section 3 of this chapter. Report and post programs can use the 4000 section for file opens.

0300–0399 Variable initialization

Define the necessary variables before main processing.

0400–0499 Define functions (DEFFNx)

Define the programmer-defined functions.

0500–0898 (see Specific Program Models later in this section)

This section is for special uses as per program task type.

0899–0999 IOLISTS

Each application may have a common, predefined IOLIST section for the entire application. This section should contain IOLISTS for all files that may be interfaced by this application, including interfaces with other applications. Provide this common section with the application in the xxLIB file, which then can be used in any new program developed for the application. This procedure eliminates the need to change IOLISTS manually for file additions. All programs in an application should have the same IOLIST section.

Applications that use string templates for file access either have no IOLIST section or have an abbreviated section with IOLISTS for other applications.

Each IOLIST line number and label must be unique in the application. It should never be different between two programs. Another application's file may use the same line number if it is not interfaced with the contending application.

Each program that accesses data files uses the IOLIST group of code (if it does not use string templates to access the file). The IOLIST code *always* resides in lines 0899–0999, which are reserved for this purpose. Do not use these lines for any other purpose, even if you don't use IOLISTS in the program. The standard IOLIST code group is built up of all the IOLISTS from all the data files that are accessed in an application. The IOLIST code placed in any program starts with this standard code group. If a program is tight on memory or does limited I/O to files, you can delete unused IOLISTS from the standard code group.

Alternate IOLISTS can be used in programs. Use ALTxxxxx: as the label format. Standardize them where possible and provide them in the application's library of common routines in the IOLIST xxFIL code section. The goal is to have one source of IOLISTS for all programs in the application.

The first line of the code, 0899, is always the signature line, which describes the contents of the IOLIST code group.

```
0899 REM 0 - (BRFIL) 0899-0999
```

Each application was originally assigned a group of 10 line numbers for its data files. Because there are now more than 10 applications, some groups of line numbers have been reused or added to. Originally the line number was allocated based on the number on which the file was opened. Now we use line labels. Each IOLIST must have a unique label. The format is [BRBAX:].

Some application functions require an auxiliary set of IOLISTs (for temporary files, for example). They can be numbered in the 990–999 range outside the usual range for the application. If you need an IOLIST from another application, use it exactly as it is in the original application's IOLIST code group. It can be moved to another line number if contention occurs. The label must be unique.

The IOLIST statement is constructed by listing each variable in the data file in the order it is stored in the data file. General practice is to name the variables in each IOLIST statement with the same first letter so that all the variables for a file can be readily identified by the first letter.

It is standard practice for keyed files to define the first variable equal to the primary key of the file. The key to the file is generally a string variable with the same starting letter as the rest of the variables in the IOLIST and a zero (for example, A0\$ for a file defined with the letter A).

The IOLIST statement may use the ALL construct for arrays. This is the IOLIST for the ARTRx file:

```
0925 ARTR: IOLIST R0$,R0,R1$,R1[ALL],R2$,R3$,R2[ALL],R3,R4$,R4,R5[ALL],R5$,R6$
```

Follow these guidelines when you use the ALL construct:

- The same technique is used for string arrays.
- The array must be dimensioned to the proper number of elements *before* using the IOLIST. DIM R1[2] is not the same as DIM R1[1:2].
- Each element of a dimensioned array must be listed in the IOLIST with no other variables between array references.
- The variables used in an application must be unique for all files in that application.

Main Processing

1000–5999 (see “Specific Program Models” later in this section)

Put DIMension statements at 1010–1099 after an ABANDON: line label and before a PAGEUP: label. This allows 'S' logic to GOTO PAGEUP and not clear out any variables. 'X' logic can GOTO ABANDON and clean up all variables.

Insert your processing code in this section. Use GOSUB and CALL to access general routines when desired.

Use top-down logic.

General Routines and Reserved Areas

<u>6000–6099</u>	<u>GENINPUT routine (see “General Routines” in chapter 3)</u>
<u>6200–6299</u>	<u>GENINFO routine (see “General Routines” in chapter 3)</u>
<u>6300–6399</u>	<u>Reserved</u>
<u>6400–6499</u>	<u>Reserved</u>
<u>6500–6699</u>	<u>Additional subroutine area</u>
<u>6700–6799</u>	<u>GENLINE routine (see “General Routines” in chapter 3)</u>
<u>6800–6999</u>	<u>Additional subroutine area</u>
<u>7000–7499</u>	<u>Reserved for future routine</u>

Text Screen

Each program that has an input screen uses the following routines to draw the screen in text mode. The code *always* resides in lines 7500–7999, which have been reserved for this purpose. These lines are used for other purposes when screen display is not done in a program. The screen display routines are made up of these three routines:

7500–7699 Print prompts (see “Coding Conventions” in this chapter)

This routine is used to display the field prompts on the text screens on a user's terminal. This section of code can be called from anywhere in the main program.

7700–7899 Print data (see “Coding Conventions” in this chapter)

This routine is used to display data on the text screens on a user's terminal. This section of code can be called from anywhere in the main program to refresh data.

7900–7999 Print boxes (see “Coding Conventions” in this chapter)

This routine is used to display the box and line graphics on the text screens on a user's terminal. It should be called *only* from the print prompts section defined above. Some programs that perform only initialization may contain just the Print Boxes routine so that the task title appears before any OPEN errors are reported. If all is okay, the next program can paint the prompts and field names.

General Routines

<u>8300–8399</u>	<u>GENOPEN routine (see “General Routines” in chapter 3)</u>
<u>8400–8499</u>	<u>GENERROR routine (see “General Routines” in chapter 3)</u>
<u>8500–8999</u>	<u>(see “Specific Program Models” later in this section)</u>
<u>9000–9199</u>	<u>GENDATE routine (see “General Routines” in chapter 3)</u>
<u>9300–9399</u>	<u>GENLOCK routine (see “General Routines” in chapter 3)</u>
<u>9400–9599</u>	<u>GENPC routine (see “General Routines” in chapter 3)</u>
<u>9600–9989</u>	<u>Reserved for use during program development</u>

Exit

<u>9990–9998</u>	<u>Exit from program - (usually RUN O1\$)</u>
<u>9999</u>	<u>END statement</u>

General Routine

<u>10000–10099</u>	<u>GENNUM routine (see “General Routines” in chapter 3)</u>
<u>10100–10299</u>	<u>GENPHONE routine (see “General Routines” in chapter 3)</u>

GUI Screen

Each program that has a GUI input screen uses the following routines. The code *always* resides in lines 17500-17899, which have been reserved for this purpose. These lines are generally not used in other program models. The GUI screen display routines are made up of these two routines:

<u>17500–17699</u>	<u>Print GUI Prompts (see “Coding Conventions” in this chapter)</u>

This routine is used to display the GUI screen on a user's workstation. This section should be accessed only from the PROMPTS (7500) section of code, and only after checking to see if the variable GUI is set.

17700–17899 ***Print GUI Data (see “Coding Conventions” in this chapter)***

This routine is used to display data on GUI screens on the user's workstation. This section should be accessed only from the DATA (7700) section of code, and only after checking to see if the variable GUI is set.

GUI Procedures

Several procedures are used to control the way in which a GUI screen is displayed and how it reacts to user input. These common routines are always found in lines 18000-18555.

18000–18099 ***Disable Fields (see “Coding Conventions” in this chapter)***

This routine enables only the key field and disables all other fields on a GUI screen. This section is usually called from the ABANDON section of code. Several routines may be used in this section to disable blocks of fields at a time depending on where the user is on the screen.

18100–18199 ***Enable Fields (see “Coding Conventions” in this chapter)***

This routine disables only the key field and enables all other fields on GUI screens. This section is usually called after the user enters a key field, which is after the PAGEUP label in many programs. This section may contain more than one routine to enable blocks of fields depending on the user's actions.

18200–18250 ***Clear Global (see “Coding Conventions” in this chapter)***

This routine is used to clear STBL(“DESTID”). On a GUI screen, if the user clicks to move from one field to another, this global variable contains the control ID that the user clicked. The program uses this ID to determine whether the program control must move forward or backward to get to the proper field. The variable's contents must be cleared when program execution is interrupted by the program, such as when field contents are invalid or when messages appear on the screen.

18400–18450 ***Destroy Screen (see “Coding Conventions” in this chapter)***

This routine is used to hide or destroy GUI Screen that are currently in use in an application. This routine can be found in programs where forms pop up on top of other forms, and need to be hidden once the user has completed the function.

18500–18555 Set up buttons (see “Coding Conventions” in this chapter)

This routine DIMensions and sets up the GUI_BUTTON\$[ALL] array. This array contains control ID’s and character keystroke equivalents for HLE screens or other screens that contain scroll regiond (primarily inquiry functions).

GUI Functions

Several functions are also provided to streamline the presentation of data in certain controls. These functions are:

18800–18850 GENCHECK (see “Coding Conventions” in this chapter)

18850–18890 GENCHECK RADIO (see “Coding Conventions” in this chapter)

18900–18950 GENLIST (see “Coding Conventions” in this chapter)

Future Space

20000–59999 Additional subroutine area (see below)

60000–64299 Reserved for future utilities

64300–64399 Used by EDIT

64400–64434 Reserved for future use

65535 Internal END statement (PRO/5 only)

Specific Program Models

Data Entry Model and Inquiry Model

Below is a description of the groups of code used in data entry programs.

Any group of code not listed below is defined in the general program model above. These programs are intensely interactive and must always use GENINPUT and GENERERROR.

<u>0400–0898</u>	<u>Extra initialization section</u>
<u>1000–3999</u>	<u>Data display, cursor positioning, data entry, edit, and verify logic</u>
<u>4000–4999</u>	<u>Write data to file, program control, chaining and/or Approve/Proceed logic</u>
<u>5000–5999</u>	<u>Subroutines for the program</u>
<u>6000–6299</u>	<u>GENINPUT routine</u>
<u>7500–7999</u>	<u>Print screen routine</u>
<u>8300–8399</u>	<u>GENOPEN routine</u>
<u>8400–8460</u>	<u>GENERERROR routine</u>
<u>9000–9199</u>	<u>GENDATE routine when needed</u>
<u>9200–9299</u>	<u>Delete record section</u>
<u>9300–9399</u>	<u>GENLOCK routine never used</u>
<u>10000–10099</u>	<u>GENNUM routine</u>
<u>10100–10299</u>	<u>GENPHONE routine</u>
<u>18800–18950</u>	<u>GUI GENxxxxxx routines</u>

Pick Screen Model

Below is a description of groups of code used in pick programs.

Any group of code not listed below is defined in the general program model above. These programs are intensely interactive and must always use GENINPUT and GENERERROR.

<u>0400–0898</u>	<u>Extra initialization area</u>
<u>1000–2999</u>	<u>Selection data display, entry, edit, and verify logic</u>
<u>3000–3999</u>	<u>Prepare for next program, call to GENOUT.PUB, mount form message, and/or forms alignment logic</u>
<u>4000–4999</u>	<u>Initialize report variables as needed and run print/post overlay program</u>
<u>5000–5999</u>	<u>Subroutines for the program</u>
<u>6000–6199</u>	<u>GENINPUT routine</u>
<u>7500–7999</u>	<u>Print screen routine</u>
<u>8300–8399</u>	<u>GENOPEN routine</u>
<u>8400–8499</u>	<u>GENERERROR routine</u>
<u>9000–9199</u>	<u>GENDATE routine when needed</u>
<u>9300–9399</u>	<u>GENLOCK routine never used</u>
<u>10000–10099</u>	<u>GENNUM routine</u>
<u>17500–17899</u>	<u>Print GUI screen routines</u>
<u>18800–18950</u>	<u>GUI GENxxxxxx routines</u>

Report Printing Model

Below is a description of groups of code used in report printing programs. This program is always preceded by a pick screen program.

Any group of code not listed below is defined in the general program model above.

<u>0400–0898</u>	<u>Extra initialization area</u>
<u>1000–1999</u>	<u>Sort routine if needed or alternate key assignment section</u>

<u>2000–2999</u>	<u>Main print loop (read file, format information, and print)</u>
<u>3000–3999</u>	<u>Print control (determine when to print break totals and headings)</u>
<u>4000–4999</u>	<u>Used as needed</u>
<u>5000–5999</u>	<u>Used as needed</u>
<u>6000–6999</u>	<u>End of File routine (print grand totals and/or End of Report just before the final form feed)</u>
<u>7000–7999</u>	<u>Print page headings</u>
<u>8000–8999</u>	<u>Used as needed</u>
<u>9000–9299</u>	<u>Used as needed</u>
<u>9300–9799</u>	<u>GENLOCK routine is installed</u>
<u>10000–10099</u>	<u>GENNUM routine</u>
<u>10100–10299</u>	<u>GENPHONE routine</u>

Combined Pick Screen and Report Printing Model

Below is a description of groups of code used in programs that perform both pick screen and report printing functions.

Any group of code not listed below is defined in the general program model above. These programs use GENERERROR during the pick phase, but switch to GENLOCK for the printing phase.

<u>0400–0898</u>	<u>Extra initialization area</u>
<u>1000–2999</u>	<u>Pick screen data entry</u>
<u>3000–3799</u>	<u>Report initialization</u>
<u>3800–3999</u>	<u>Select output destination</u>
<u>4000–4999</u>	<u>Resolve ranges</u>

<u>5000–5999</u>	<u>Print logic (including SETESC 9350, SETERR GENLOCK) and end of file routine</u>
<u>6000–6300</u>	<u>GENINPUT routine</u>
<u>7000–7499</u>	<u>Print page headings</u>
<u>8300–8399</u>	<u>GENOPEN routine</u>
<u>8400–8499</u>	<u>GENERERROR routine</u>
<u>9000–9199</u>	<u>GENDATE routine when needed</u>
<u>9300–9399</u>	<u>GENLOCK routine</u>
<u>10000–10099</u>	<u>GENNUM routine</u>
<u>10100–10299</u>	<u>GENPHONE routine</u>
<u>18800–18950</u>	<u>GUI GENxxxxxx routines</u>

Posting Model

Below is a description of groups of code used in posting programs. This program type is always preceded by a pick screen program.

Any group of code not listed below is defined in the general program model above.

<u>0400–0898</u>	<u>Extra initialization area</u>
<u>1000–1999</u>	<u>Sort routine if needed or alternate key assignment section</u>
<u>2000–2999</u>	<u>Main post loop (read files, format information, and write to other files)</u>
<u>3000–3999</u>	<u>Additional processing logic if needed</u>
<u>4000–4999</u>	<u>Program control statements (determine when to exit or start next process)</u>

<u>5000–6999</u>	<u>Used as needed</u>
<u>7000–7999</u>	<u>Print page headings</u>
<u>8000–8999</u>	<u>Used as needed</u>
<u>9000–9299</u>	<u>Used as needed</u>
<u>9300–9799</u>	<u>GENLOCK routine is installed</u>
<u>10000–10099</u>	<u>GENNUM routine if a log is produced</u>

Horizontal Line Entry Initialization Model (?????INI)

Below is a description of groups of code used in horizontal line entry (HLE) initialization programs. This program type is always followed by an HLE control program.

Any group of code not listed below is defined in the general program model above.

<u>0400–0898</u>	<u>Extra initialization area</u>
<u>1000–5999</u>	<u>Variable initialization, sort file creation (if required), and header field entry</u>
<u>7500–7999</u>	<u>Print screen routine</u>
<u>10000–10099</u>	<u>GENNUM routine</u>
<u>17500–17899</u>	<u>Print GUI screen routines</u>
<u>18000–18550</u>	<u>GUI procedures</u>
<u>18800–18950</u>	<u>GUI GENxxxxxx routines</u>

Horizontal Line Entry Line Item Entry Model (?????LN?)

The groups of code used in horizontal line entry (HLE) control programs are similar to the standard data entry model. This program type always follows an HLE control program. This program controls the editing of line items and returns to the control program.

Horizontal Line Entry Control Model (?????CON)

Below is a description of groups of code used in horizontal line entry (HLE) control programs. This program type may follow an initialization program or may be run directly from the menu. This program controls all line commands and may also control header and/or line item entry.

Any group of code not listed below is defined in the general program model above.

<u>0400–0898</u>	<u>Extra initialization area</u>
<u>1000–1099</u>	<u>Startup and Abandon logic</u>
<u>1400–1549</u>	<u>Variable initialization for HLE</u>
<u>1550–1799</u>	<u>HLE Command display and control</u>
<u>1800–2399</u>	<u>Edit key processing (Up, Down, Page Up, Page Down, Home and End)</u>
<u>2400–2699</u>	<u>HLE screen display and HLE line build</u>
<u>2700–2999</u>	<u>Get last line number routine and go to line number routine</u>
<u>3000–3499</u>	<u>Startup and header logic</u>
<u>3500–3599</u>	<u>Append line logic</u>
<u>3600–3899</u>	<u>Edit line and line item entry logic</u>
<u>3900–3999</u>	<u>File write logic</u>
<u>4000–4999</u>	<u>Insert line and delete line logic</u>

<u>5000–5999</u>	<u><i>Routine-specific code (including scratch file preparation logic, if necessary)</i></u>
<u>6000–6999</u>	<u><i>GENINPUT and GENLINE</i></u>
<u>7500–7999</u>	<u><i>Screen display</i></u>
<u>8000–9899</u>	<u><i>Used as needed</i></u>
<u>10000–10099</u>	<u><i>GENNUM routine</i></u>
<u>17500–17899</u>	<u><i>Print GUI screen routines</i></u>
<u>18000–18550</u>	<u><i>GUI procedures</i></u>
<u>18800–18950</u>	<u><i>GUI GENxxxxxx routines</i></u>

Program Chains and Overlays

A program chain is a series of programs working together to perform a common task while each program performs a particular task.

Each program in a chain should have the same first four to seven characters of its name. The last one to three characters identify the individual programs.

The first program of the chain is the only one that is invoked from the menu.

Public Programs

Public programs are sections of code that have been isolated into separate programs and are CALLED into system memory to be executed. The CALLing program remains in main memory, waiting for the public program to return execution to it.

You must maintain compatibility with the OSAS initial standards, which have been well-observed by both Open Systems and independent software developers.

These routines can be accessed from any application program. These subroutines are accessed with the CALL statement instead of a RUN or a GOSUB. Programs can access these external subroutines without loading the subroutine and then reloading the main program.

Public programs allow for local variables and parameter passing. The ENTER statement is used to define which variables are call-by-value, call-by-reference or global. This allows for well-isolated routines that affect the main program in a predictable manner. Global passing (ENTER with no parameters) can be very slow, because BB^x creates duplicate pointers in the public program for all data.

The public program always EXITs to the statement following the CALL statement. The EXIT may carry an error value. This should be handled by the main program.

Public programs use the SETERR and SETESC verbs. Errors *must* be trapped in public programs. Public programs do not nicely report errors to the user. The errors must be trapped and then acted upon or returned to the CALLing program. The user cannot escape to BASIC from a public program, but a programmer can by setting SETOPTS \$08\$.

Public programs have the following advantages over general routines:

- Code modification. You can change a single public program, rather than having to merge a new general routine into every program.
- Code size. Only one copy of the public program is kept on the disk, rather than each of the programs containing the general routine.
- Memory usage. The public programs are loaded outside the workspace(s) for the main program and data, allowing more code to be resident for the current task. This may help reduce the number of pauses that occur when new programs are loaded.
- Load time. Public programs can be ADDRed to resident memory, allowing them to be accessed many times with only one load.

ADDR and DROP

A nice feature of PRO/5 programs is that they can be loaded and kept in memory so that the interpreter doesn't have to load the program each time it is CALLED or RUN. This process is called ADDRing, adding to resident memory.

When a public program is going to be CALLED repeatedly, the program should be ADDRed. These programs are loaded by the ADDR verb and can be called at any time. The interpreter will know that the program has been loaded into memory and will not load it again. CALLing memory-resident procedures may save time.

When a public program is going to be CALLED only occasionally, the program may be loaded by the CALL statement. These programs are loaded at the time of CALL. It takes seconds to load and begin the routine.

At system startup, the RMSETUP program ADDRs a list of public programs. The menu program drops all public programs except those specified in an internal list. This task is executed each time an application returns to the menu. The internal list includes GENERERROR.PUB, GENINPUT.PUB, GENDATE.PUB and GENOPEN.PUB. In GUI mode, GENGUINP.PUB, GENGURES.PUB, GENGUFNC.PUB, and GENGULIB.PUB are also ADDRed into memory.

General Routine Structure

General routines are sections of code that are merged into each program when needed. They provide a consistent method of performing particular tasks so that the user can expect a consistent response to a similar condition regardless of the application. They also eliminate the need to enter the same section of code several times.

You must maintain compatibility with the OSAS standards. Future standards will try to retain as much of the current functionality and entry points as possible so that a great deal of code does not have to be changed when a module is improved.

You can treat these code modules as black boxes. A set of input and output data is defined for each OSAS system routine. The functionality of each routine is fixed to perform only particular tasks. These routines must never be changed except by OSAS development for new software versions.

General routines are internal to each program. These subroutines are accessed with the GOSUB statement.

General routines have well-defined entry points.

The general routines most often exit with a RETURN statement, transferring control to the statement following the GOSUB statement. The general routine may allow execution to exit in different ways. A routine may allow the user to enter a command during an error message and force a restart instead of returning to the place where the routine was executed.

General routines have the following advantages:

- Code consistency. One set of code is merged into all programs that need the task performed.
- User consistency. The user sees the same routine executed so that the functionality remains identical between tasks.

For a routine to qualify as a general routine, it must be used extensively in most or all of the programs in all the applications. The routine must not have any code specific to any application. For example, the INUPD routine is not considered a general routine because it contains code specific to the Inventory application. The routine does allow for a consistent interface with the Inventory application, however, so it is defined as part of the Inventory application standards. By contrast, GENERERROR is used in all applications in the same manner to trap particular error conditions for consistent handling and user notification.

All general and local routines can be merged directly into programs, using the PRO/5 _update function and the library files.

General routines sometimes have auxiliary code sections that work with them. You must define these sections for the particular task programs. These sections include 9200 delete record, 8470 abandon (X) and start-over (S) logic processing, and 9990 exit from program, among others.

WARNING: You must use the general routines as they are supplied; do not change them.

EXCEPTION: Some routines are absolute. The INFO routine at 6200 is an example. GENLIB contains a common do-nothing GENINFO section that allows GENINPUT to pass control when the INFO key is pressed for all programs. Only programs that contain an activated xxINFO section should perform a task. This routine is not necessarily fixed even within a single application. It may be desirable in some programs to present a single task in lieu of the full INFO menu. It is permissible to modify this section in each program to customize the functionality.

Coding Conventions

This section describes many of the common coding conventions used throughout OSAS programs: screen display, data entry, command processing, error messages, and file I/O. The examples are common conventions and do not show all usages of the general routines. See the General Routines section for the complete usage of particular general routines.

Screen Display

Text Screen Prompts

Field prompts are short descriptors that identify the purpose of each value. They should be short and meaningful. Do not abbreviate unless absolutely necessary. Begin each word, except articles and prepositions of less than four characters, with a capital letter. The color for these data prompts in text mode is in the global strings "FTXT", "FTXT2" and "FTXT3" and should be displayed in background ('SB').

Field prompts are displayed in the 7500 section. Do not display prompts from anywhere else. Go to this section at the beginning of each program before any file opens. The section must include the command GOSUB BOX. Because data cannot be entered outside predefined positions, this section usually needs to be used only once per screen.

This routine resides in lines 7500–7699. Use the following lines of code as the starting point of the routine. Place additional code in lines 7525–7699.

```
7500 REM 7500,5 "Display Screen"
7505 PROMPTS:
7510 IF GUI THEN GOTO GUI_PROMPTS
7515 GOSUB BOX
7520 PRINT STBL("FTXT"), 'SB',
.
.
.
7699 RETURN
```

The routine performs the following operations:

- if the user is in GUI mode, go to GUI_PROMPTS section to paint the screen
- call the Screen Box Paint routine (see below)
- change the color for field names
- set up background intensity
- display the field names

These are the conventions for constructing the code for this routine:

- Print lines default output to the screen. Do *not* use a channel number.
- End print lines with a comma.
- Position with direct cursor addressing, @(c,r). Use both row and column numbers. If you print at more than one position on the same line, the additional items can be printed using column positioning only.
- Keep variable assignment statements to a minimum.
- Do not change precision. You could unintentionally alter the precision of other data. PRO/5 correctly displays masked numeric values above the current precision.
- Print the fields on a line from left to right and from top to bottom. Do *not* jump around to various print positions, and do not use any mnemonics except direct cursor addressing.

This routine should be called before any other operations in each program. The screen should be displayed before any file opens.

GUI Screen Prompts

For GUI Screens, all prompts are stored in resource files called BRC files. These files store all prompt information as individual controls. See the later sections of this chapter for more information about the standards for individual controls.

GUI screens are displayed using the 17500 section. Do not load screens from anywhere else. You must branch to this section of code from the text-based PROMPTS section if the user is in GUI mode. Because there is no box to display, there is no need for a GUI equivalent to the text-based BOX section of code.

This routine resides in lines 17500-17699. Use the following lines of code as the starting point of the routine. Place additional code in lines 17510-17689 as needed.

```
17500 REM 17500 "Display Screen"
17505 GUI_PROMPTS:
.
.
17690 PRINT (OSGUI.CHAN) 'TITLE' (0,TITLE$), 'SHOW' (0)
17699 RETURN
```

This routine performs these operations:

- loads up the GUI screen
- displays the GUI menu bar
- enables, disables, hides, or shows certain controls where necessary
- sets the title and displays the entire screen

There are several different examples, each varying in style on how to load screens. The most common example is shown below:

```
17500 REM 17500 "Display Screen"
17505 GUI_PROMPTS:
17510 LET OSGUI.SCRNID$="ARPRR.BRC",OSGUI.WIND_ID[1]=1,OSGUI.CONTEXT_POINTER=1
17515 LET MSG$=" "; CALL "GENGURES.PUB",FILE[ALL],OSGUI$,MSG$; IF
17515:CVS(MSG$,3)<>" " THEN LET X9$=MSG$,07=1; GOSUB GENERROR; GOTO QUIT
17520 CALL "GENGUFNC.PUB::BUILD_MENU_STRINGS",MENUHLP$,MENU1$
17525 LET MENU1$=MENU1$+MENUHLP$; PRINT (OSGUI.CHAN) 'SETMENU',MENU1$; LET
17525:MENU1$=" ",MENUHLP$=" "
17690 PRINT (OSGUI.CHAN) 'TITLE' (0,TITLE$), 'SHOW' (0)
17699 RETURN
```

This section of code performs these tasks:

- Sets the BRC name to ARPRR.BRC, sets the window number to get from the resource file to 1, and sets the context pointer to 1 (because there is only one screen to load). The number in OSGUI.CONTEXT_POINTER must be the same as the element of the OSGUI.WIND_ID[ALL] array with which you are currently working.
- Calls GENGURES.PUB to retrieve the screen from the resource file.
- Calls GENGUFNC.PUB to build the standard menu sections for GUI screens. These strings contain the Commands, Edit, Modes, Other, and Help pull-down menu functions.
- Creates the menu on the current form.
- Sets the title of the current form and displays the form.

It is important to note that GENGURES.PUB retrieves the screen and displays it. Because all standard OSAS screens are set to be initially invisible, you will not see the screen until the 'SHOW' mnemonic is executed. Until then, you can manipulate the form without the user seeing any changes.

The following example of loading GUI screens handles screens that have a scroll region, or HLE (horizontal line entry) screens:

```

17500 REM 17500 "Display Screen"
17510 GUI_PROMPTS:
17515 IF TASK$<>" " THEN PRINT (OSGUI.CHAN)'HIDE'(0),'CONTEXT'
17515:(OSGUI.CUR_CONTEXT[1]) ; LET OSGUI.CONTEXT_POINTER=1; RETURN
17520 LET OSGUI.SCRNID$="ARCSH.BRC",OSGUI.WIND_ID[1]=1,OSGUI.CONTEXT_POINTER=1
17525 LET MSG$=" "; CALL "GENGURES.PUB",FILE[ALL],OSGUI$,MSG$; IF
17525:CVS(MSG$,3)<>" " THEN LET X9$=MSG$,O7=1; GOSUB GENERERROR; GOTO QUIT
17530 CALL "GENGUFNC.PUB::BUILD_MENU_STRINGS",MENUHLP$,MENU1$
17535 CALL "GENGUFNC.PUB::BUILD_HLE_MENU",MENUHLE$
17540 CALL "GENINFO.PUB::GET_GUI_INFO_MENU",MENUINFO$,O2$
17545 LET MENU$=MENU1$+MENUHLE$+MENUINFO$+MENUHLP$; PRINT
17545:(OSGUI.CHAN)'SETMENU',MENU$
17550 IF O2$(10,1)="1" THEN PRINT (OSGUI.CHAN)'SHOW'(1000,4000,14000),
17550:'ENABLE'(1000,4000,14000)
17605 PRINT (OSGUI.CHAN)'TITLE'(0,TITLE$),'SHOW'(0)
17610 RETURN

```

This section of code (which is executed in OSAS when you return to the Cash Receipts screen from the Edit Cash Receipts screen):

- Checks TASK\$, and if it is null, hides the form and sets the current context.

- Sets the BRC name to ARCSH.BRC, sets the window number to get from the resource file to 1, and sets the context pointer to 1 (again, there is only one screen to load).
- Calls GENGURES.PUB to retrieve the screen from the resource file.
- Calls GENGUFNC.PUB to build both the standard menu sections for GUI screens and also the scroll commands menu for use on HLE screens only.
- Calls GENINFO.PUB to build the Information menu. In character mode this is the Shift-F2 menu.
- Creates the menu on the current form. Notice that the individual menu strings are not cleared so that they can be reused later in the code without having to call GENGUFNC.PUB or GENINFO.PUB again.
- Checks to see if batching is used and either hides or shows the batch ID prompt and entry fields, as necessary.
- Sets the title of the current form and displays the form.

There are other programs that display screens on top of other screens (for example, daily work entry chains), or that use a chain of screens shown one at a time (for example, customer or vendor maintenance). Here is an example of the code used in loading such screens:

```

17500 REM 17500 "Display Screen"
17510 GUI_PROMPTS:
17520 IF ARENT_HEAD_CONTEXT THEN PRINT (OSGUI.CHAN) 'CONTEXT'
17520:(OSGUI.CUR_CONTEXT[ARENT_HEAD_CONTEXT]); LET
17520:OSGUI.CONTEXT_POINTER=ARENT_HEAD_CONTEXT; GOTO 17555
17525 LET OSGUI.SCRNID$="ARENT.BRC"; FOR I=1 TO NUM(STBL("GUIWIND")); IF
17525:OSGUI.WIND_ID[I]=0 THEN LET OSGUI.WIND_ID[I]=2,OSGUI.BASE_CONTEXT[I]=
17525:-1,OSGUI.CONTEXT_POINTER=I,ARENT_HEAD_CONTEXT=I; EXITTO 17530 FI; NEXT I
17530 CALL "GENGURES.PUB",FILE[ALL],OSGUI$,MSG$; IF CVS(MSG$,3)<>" " THEN LET
17530:X9$=MSG$,O7=1; GOSUB GENERERROR; GOTO QUIT
17535 LET OSGUI.BASE_CONTEXT[I]=OSGUI.BASE_CONTEXT[1]
17540 LET MENU$=MENU1$+MENUINFO$+MENUHLP$; PRINT (OSGUI.CHAN) 'SETMENU',MENU$
17545 IF O2$(2,1)="1" THEN PRINT (OSGUI.CHAN) 'SHOW'(14020),'ENABLE'(14020)
17550 IF O2$(10,1)="1" THEN PRINT (OSGUI.CHAN) 'SHOW'(1000,4000,14000)
17555 PRINT (OSGUI.CHAN) 'TITLE'(0," Header Information "),'SHOW'(0)
17560 RETURN

```

This section of code performs these tasks:

- Checks ARENT_HEAD_CONTEXT and, if it is set, changes the context to that array element. This variable is set once the form is loaded (see line 17525). The routine also skips the next form load because it is already loaded.
- Sets the BRC name to ARENT.BRC and searches the OSGUI.WIND_ID[ALL] array for an unused element that can be used for the header screen. The unused element is then set to the window ID. The routine also sets OSGUI.BASE_CONTEXT[] to -1 because there is no status bar on this form. Then the routine sets ARENT_HEAD_CONTEXT to the value of the loop counter, in this case, I.
- Calls GENGURES.PUB to load the form and then shows or hides controls based on option settings (Interface to Inventory and Batching in this example).

On forms that need to hide other forms before they are displayed, use the 'HIDE' mnemonic at the top of this section of code to hide other forms, then check the context (as in the example on line 17520) to switch to or load up the next screen.

Character Data Display

Data should be displayed only in the 7700 section. Keep manual field repainting to a minimum inside the main processing section of code. The color for the data fields is kept in the global strings "FTXT", "FTXT2" and "FTXT3" and should be displayed in foreground ('SF').

The 7700 section contains the print information for the data to be painted on the screen. Use this section to refresh the screen data when needed. Use temporary variables to display data sparingly. You should display the same variables that will be written to the file unless a translation is required.

This routine resides in lines 7700–7899. The following lines of code should be used as the starting point of the routine. Place additional code in lines 7710–7889.

```
7700 REM 7700 "Display Data"
7701 DATA:
7704 PRINT 'CF', 'SF', STBL("FTXT"),
.
.
.
```

7890 RETURN

The routine performs the following operations:

- clears data displayed in foreground mode
- changes the color to foreground text color for the field names
- displays the data fields

These conventions are used in constructing the code for this routine:

- Print lines default output to the screen. Do *not* use a channel number.
- End print lines with a comma.
- Position with direct cursor addressing, @(c,r). Use direct row and column positioning unless you are printing multiple items on the same line.
- Keep variable assignment statements to a minimum.
- Do not change precision.

PRO/5 correctly displays masked numeric values regardless of the internal value. If you use a PRECISION statement and perform a calculation, you might inadvertently alter the precision of the data you are displaying. If you need to perform a multiplication or division operation on a variable before display, use the ROUND() function to round the calculation appropriately, regardless of the current precision setting. This does not affect the precision of other calculations in the program.

Precision setting is required only when you want to force internal stored data to have a specific decimal size. The displayed value is rounded off when printing with a mask, and it does not affect the internal value in memory. You need to use PRECISION only to format the output of a multiplication or division calculation.

- Print the fields on a line from left to right and top to bottom. Do *not* jump around to various print positions.
- Do not use mnemonics except direct cursor addressing.

- Use the FNNUM\$(x,x\$) function, rather than the x:x\$ construct, to print numeric variables. This function traps mask errors automatically, and is present in any program in which the GENNUM routine has been included.
- For phone numbers, use the FNPHONE function to display the phone number properly. You must make sure your phone number fields are using 20 characters, with the mask coming from the Country file.

GUI Data Display

Data should be displayed only in the 17700 section. Keep manual field repainting to a minimum inside the main processing section of code.

The 17700 section contains the print information for the data to be displayed in the fields on the screen. Use this section to refresh the data as needed. Use temporary variables to display data sparingly. You should display the same variables that will be written to the file unless a translation is required.

This routine resides in lines 17700–17899. The lines of code below should be used as the starting point of the routine. Place additional code in lines 17710–17889.

```
17700 REM 17700 "Display Data"
17705 GUI_DATA:
17710
.
.
.
17890 RETURN
```

Because all data should be displayed on the screen, and printing any data to any INPUTE or INPUTN controls will clear those controls of the previous data, there is no need to clear all controls before printing data.

When you print data in an INPUTN control, you must use the STR() function on the variable, removing leading and trailing spaces, when you print to the control. The INPUTN control handles the mask formatting automatically.

The rules of precision are the same as in character screens. However, where the FNNUM function is used on character screens for fields in which the numeric mask can vary, you must use the FNGUINUM function for GUI controls. The FNGUINUM function sets the proper mask for numeric GUI controls.

Use the FNGUIPHONEMASK function to display phone numbers on graphical screens. This function sets the proper mask for phone number controls.

Other functions are provided to assist in displaying data to screens, including

- FNCHECK, which selects or clears a GUI check box
- FNCHECKBOX_RADIO, which selects or clears radio buttons
- FNLIST, which selects a value from a list button control

Display Character Boxes and Lines

Each text screen in OSAS is logically boxed to break up the fields on the screen in a logical order. The general boxing routine is always coded within line numbers 7900–7999. See the definition of this routine below.

```
7900 REM 7900 "Display Boxes"
7901 BOX:
7905 PRINT 'BACKGR',STBL("FBKG"),STBL("FBX"),'DEFAULT','CS','SB',
7910 PRINT 'BOX'(0,0,79,24),'GS',
7920 PRINT @(0,21),"6",'BOX'(1,21,77,0),@(78,21),"7",
7930 PRINT @(xpos,ypos),"6",'BOX'(xpos+1,ypos,xlen,0),@(xlen+1,ypos),"7",
7940 PRINT @(xpos,ypos),"9",'BOX'(xpos,ypos+1,0,ylen),@(xpos,ypos+ylen+1),"8",
7940:'GE'
7985 PRINT 'SB',STBL("FTTL"),@(INT((79-LEN(TITLE$))/2),0),TITLE$,
7995 RETURN
```

7905 *Clears the screen, sets background, and sets up the color for the box.*

7910 *Prints the surrounding box and turns graphics on for the lines that follow.*

7920 *Prints the line on line 21 that commonly appears on input screens.*

7930 *Prints a horizontal line with left and right tees. This line is duplicated for each horizontal line.*

xpos *x-coordinate of the starting position of the line—where a line intersects a vertical line*

ypos *y-coordinate of the line*

xlen *length of the line without the tees*

7940 *Prints a vertical line with top and bottom tees. This line is duplicated for each vertical line.*

xpos *x-coordinate of the line*

ypos *y-coordinate of the starting position of the line—
where a line intersects a horizontal line*

xlen *length of the line without the tees*

7985 *Prints the title of the program.*

The last line must include the 'GE' mnemonic to turn off graphics (see line 7940 above). Put commas at the end of each line to hold the cursor and speed up printing.

Below is a table of common line symbols and the corresponding characters to print in graphics mode to produce them. Add lines to the box as desired for your task.

Line Symbol	Character
=	0
	1
┌	2
┐	3
└	4
┘	5
├	6
┤	7
┼	8
┴	9
┴	:

Data Entry

Field Input

Use the general input routine, GENINPUT, for all data entry. Below are the guidelines for the format of the GOSUB statement and related variable assignments.

Record Key Input

```
1200 LET X0=1,X1=10,X3=0,X4=6,X9=1,WINDID$="ARACC W1",GUI_ID=4000
1210 GOSUB GENINPUT; REM "Customer ID"
```

WINDID\$ specifies that an inquiry window should be available for this field. X9=1 clears X\$ and stops Y\$="P" or Y\$="J". GUI_ID is the control ID of the field in the BRC file for this screen.

Normal String Input

```
1200 LET X0=1,X1=10,X3=0,X4=30,X$=C1$,GUI_ID=4000
1210 GOSUB GENINPUT; REM "Customer Name"
```

C1\$ is the default customer name.

Masked String Input (numeric characters only)

```
1200 LET X0=1,X1=10,X3=2,X1$=M2$,X4=LEN(X1$),X$=FNS(R9$),GUI_ID=4000
1210 GOSUB GENINPUT; REM "Social Security Number"
```

M2\$ is the social security number mask and is preset to " - - ". FNS(R9\$) is a programmer-defined function that formats a social security number stored in R9\$.

Numeric Input

```
1200 LET X0=9,X1=19,X3=1,X1$=M1$,X4=LEN(X1$),X=C3,GUI_ID=4000
1210 GOSUB GENINPUT; REM "Credit Limit"
```

M1\$ is the numeric mask and is preset to "#####0" in this example.

String Input Forced to Uppercase

```
1200 LET X0=10,X1=45,X3=3,X4=2,X$=S1$,GUI_ID=4050
1210 GOSUB GENINPUT; REM "State"
```

Masked String Input (any alphanumeric characters)

```
1200 LET X0=10,X1=45,X3=4,X1$=M3$,X4=LEN(X1$),X$=SN$,GUI_ID=4000
1210 GOSUB GENINPUT: REM "Serial Number Field"
```

M3\$ is an alphabetic mask and is preset to "SN - -]" in this example.

YES/NO Toggled Input

```
1200 LET X0=9,X1=19,X3=5,X4=3,X1$="",X$=YN$,GUI_ID=4000
1210 GOSUB GENINPUT: REM "Do You Want to..."
```

M1\$ is set to null to indicate a YES/NO toggle. Setting X4=3 tells GENINPUT to toggle the words *YES* and *NO*; X4=1 toggles the letters *Y* and *N*.

Other Toggled Input

```
1200 LET X0=9,X1=19,X3=5,X4=5,X1$="Red WhiteBlue GreenbLack",X$=COLOR$,
1200:GUI_ID=4000
1210 GOSUB GENINPUT: REM "Color"
```

M1\$ is a string containing the possible toggle values. GENINPUT parses the X1\$ string in increments of X4 to define the valid entries. The first capitalized character of each valid entry is the activation key for the toggle. In the example the valid entries are R, W, B, G and L (for black). If the first character of an entry is a space and no capitalized letter is included, space is a valid entry.

If capital letters cannot be used as the first letter of a word in X1\$, you may set X3=15. This will cause GENINPUT to search for the first capitalized letter within each section of X1\$ and use that letter for the toggle key.

Date Input

```
1200 LET X0=9,X1=19,X3=6,X=N1[1],GUI_ID=4000
1210 GOSUB GENINPUT: REM "Invoice Date"
```

N1[1] is a Julian date. GENINPUT uses the input mask " / / " and the input length of 10 automatically when X3=6.

Full Command Input

```
1200 LET X0=10,X1=5,X3=-9,X$="",GUI_ID=4999
1210 GOSUB GENINPUT: REM "Command entry"
```

GENINPUT will return these commands in Y\$ in addition to the standard letter commands defined later in this section:

+	Down Arrow
!	Return Key
<	Left Arrow
>	Right Arrow
(	Home Key
)	End Key
=	Backspace Key
^	Insert Key
*	Local window processing
#	Local maintenance processing or Delete Key
Z	Escape Key

The programmer must interpret and respond to the command returned in Y\$ in the main program.

In GUI screens, you must make sure that an invisible GRID control is placed at the end of the screen in order to the proper values for X and Y back. See the previous sections for detailed information on GUI screen standards.

String Input (with the cursor positioned at the end of the line)

```
1200 LET X0=9,X1=19,X3=7,X4=64,X$=FILENAME$,GUI_ID=4000
1210 GOSUB GENINPUT; REM "Output Filename"
```

FILENAME\$ contains a default path name for the output file. GENINPUT displays the path name and positions the cursor after it for additional input.

GL Period Input

```
1200 LET X0=9,X1=19,X3=10,X=N1[2],GUI_ID=4000
```

```
1210 GOSUB GENINPUT; REM "GL Period"
```

N1[2] is a GL period. GENINPUT automatically uses the mask "#0" and the length of 2 when X3=10. Entry is limited to periods defined in the CNVTxxx table in RM.

Year Input

```
1200 LET X0=9,X1=19,X3=11,X=N1[3],GUI_ID=4000
1210 GOSUB GENINPUT; REM "Year"
```

N1[3] is a valid year. GENINPUT uses the input mask "###0" and the input length of 4 automatically when X3=11. Entry is limited to valid years (or blank), and 2-digit years are converted to 4-digit years automatically.

Period/Year Input

```
1200 LET X0=9,X1=19,X3=12,X$=STR(N1[2]:"#0")+ "/" +STR(N1[3]:"0000"),GUI_ID=4000
1210 GOSUB GENINPUT; REM "Period/Year"
```

X\$ is a valid period/year. GENINPUT uses the input mask " / " and the input length of 7 automatically when X3=12. Entry is limited to valid periods and years (or blank), and 2-digit years are converted to 4-digit years automatically. Input behaves in the same manner as date input (using the +, -, and D keys).

GL Account Input

```
1200 LET X0=9,X1=19,X3=13,X$=G0$(2,12),GUI_ID=4000
1210 GOSUB GENINPUT; REM "GL Account"
```

X\$ is a valid GL account. GENINPUT uses an input mask corresponding to the GL mask defined if GL is installed for the company. The mask and length of the field are assigned automatically when X3=13. Entries are zero-filled to the right.

Uppercase String Input (with the cursor at the end of the line)

```
1200 LET X0=9,X1=19,X3=7,X4=64,X$=FILENAME$,GUI_ID=4000
1210 GOSUB GENINPUT; REM "Output Filename"
```

FILENAME\$ contains a default path name for the output file. GENINPUT displays the path name and puts the cursor after it. The input is also forced to uppercase.

Additional GENINPUT Rules

Use the following additional style rules to create the variable assignments for the call to GENINPUT:

- Define the variables in the order listed above.
- Define the default field value as the next to last variable on the assignment line. Define GUI_ID as the last variable on the assignment line.
- You should either use line labels and remark statements before the assignment line to clarify what field will be prompted for, or put a remark at the end of GOSUB GENINPUT that contains the field name being entered.
- Define *all* the assignments for each input field before each GOSUB GENINPUT. The Up and Jump commands may make the program skip before variable assignments that were set at other input field areas.
- No other variable assignments should be performed on the same line of code.
- Define the numeric print mask via a variable. Define masks at the top of the first program of a task chain. This variable can then be used throughout the program.
- Use the LEN(X1\$) function to define the maximum length of input variable X4. This makes the code independent of the actual value of the display mask.
- Forced fields, where X9=1, do not need to set a default value of X or X\$, because the variable will be cleared.
- If the field shares a help screen with other similar fields in the application, you can set the variable FLD_HLP\$ to a help screen name.

Commands Processing

The GENINPUT and GENERERROR routines can return user commands in the variable Y\$. The commands that most programs need to handle are summarized below.

Command	Description
D	Delete a record
J	Jump to the next data group
M	Exit to the menu
P	Proceed - save a record
S	Start over
U	Move up to the previous field
X	Abandon the current record
I	Invoke information windows
B	Programmer-defined command
C	Programmer-defined command
G	Programmer-defined command
K	Programmer-defined command
N	Programmer-defined command
Q	Programmer-defined command
R	Programmer-defined command

Some of the programmer-defined commands are used in some OSAS programs. For example, the **B** command (normally **Shift+F3**) is used for General Ledger drill-down, Inventory Item lookups, sales tax breakdowns and generating purchase requisitions in Sales Order.

Below are the coding conventions for the commands.

Command**Convention****D - Delete**

GENINPUT performs a GOSUB 9200 when the **Delete** command is issued (usually by pressing the **F3** key). The **Delete** command is implemented by adding code to the 9200 code section.

The routine at line 9200 looks like this when **Delete** is not available:

```
9200 REM "DELETE NOT IMPLEMENTED"  
9210 RETURN
```

Set YDISABLE\$="D" in the program initialization when **Delete** is not available. The RETURN verb ensures that control is passed back to the calling field if YDISABLE\$ is not set.

Within an HLE program, if **Delete** is not available while in the scroll region, set DISABLE\$="D".

To accomplish a **Delete** task in a program, use code similar to this example:

```
9200 REM 9200 "DELETE ROUTINE"  
9210 LET O7=3; GOSUB GENERERROR  
9220 IF Y$<>"D" THEN RETURN  
9230 REMOVE (FILE[3],KEY=X0$,DOM=9240)  
9240 LET Y$="X"  
9250 GOTO 8452
```

Always simulate the **Abandon (X)** command after **Delete**. If the **Delete** command is not verified, the routine will return to the data field where the first **Delete** command was issued.

Command**Convention****J - Jump**

The **Jump** command returns control to the main program. For the **Jump** command to be executed properly, you must add the following code to the beginning of each group of data fields:

```
1405 IF Y$="J" THEN LET Y$=""
```

As long as Y\$ is "J", GENINPUT will simply return without performing data entry. The main program executes statements until Y\$ is cleared. In this way data fields that require error checking will step through the error checking statements.

When this extra processing is not desired, place the following statement in the appropriate places:

```
1205 IF Y$="J" THEN GOTO 1405
```

Statement 1405 should clear Y\$ if Y\$ is "J".

M - Exit

GENERERROR (from GENINPUT) performs a GOTO 9990 when the **Exit** key is pressed (and verified if necessary). This is the convention for the 9990 section:

```
9900 REM 0 "Return to Menu"
9991 QUIT:
.
.
.
9998 RUN 01$
9999 END
```

Additional lines can be inserted at 9990 if the program has special processing to perform before exiting.

NOTE: The **Exit** command takes you back one level from where you are. In most functions, this usually means going to the menu. Some screen overlay functions require a local **Exit** command to provide the user more control.

Command	Convention
P - Proceed	The Proceed (PgDn) command returns control to the main program. As with the Jump command, GENINPUT does not perform data entry while Y\$ is "P". If necessary, code can be skipped by placing the following line in the appropriate location(s): <pre>1205 IF Y\$="P" THEN GOTO 2000</pre> Some fields may require user interaction. You can use <code>1205 IF Y\$="P" THEN LET Y\$=" "</code> to stop execution and force the user to enter a required field. After all data fields have been entered, the following code should be executed before the record is written: <pre>2000 REM "Save the record" 2005 PROCEED: 2010 LET O7=2; GOSUB CLEAR_GLOBAL; GOSUB GENERROR 2020 IF Y\$="U" THEN GOTO LAST 2030 IF Y\$<>"P" THEN GOTO PROCEED</pre> This code forces the user to press the Proceed key before the record is written if the option in Workstation Defaults for verification is set to ON.
S - Start Over	The Start Over command performs a GOTO 8480. A RESET is also performed to clear GOSUB/FOR information if it exists. The function of Start Over is to move the cursor to the first nonkey field on the screen without erasing the displayed data. A program should have a line labelled PAGEUP: and the appropriate GOTO statement at line 8480, such as: <pre>8480 GOTO PAGEUP</pre> The above code assumes that the Start Over key cannot be pressed at the key field(s). This is accomplished by setting YDISABLE\$ to "S" at the key field(s), and then clearing YDISABLE\$ at the first nonkey field. If YDISABLE\$ isn't used, the 8480 line needs additional logic for the GOTO statement when key fields are involved. If the screen contains simple X8\$ help messages, add <code>X8\$=" "</code>; <code>GOSUB GENERROR</code> here to clear them.

Command**Convention****U - Up**

The **Up** command is always available in programs that have the GENINPUT routine. To implement it, place the code immediately after GOSUB GENINPUT. It should look like this:

```
1402 IF Y$="U" THEN GOTO PREVIOUS
```

Without this additional code, the program will move down one field instead of up one field when the user uses the **Up** command.

A program should allow the **Up** command to move only from a key field to the previous key field or from a nonkey field to the previous nonkey field. The **Up** command at the first key or nonkey field should stay at that field.

X - Abandon

The **Abandon** command (verified if necessary) performs a GOTO 8470. A RESET is also performed to clear GOSUB/FOR stack information if it exists. The function of the **Abandon** command is to clear the data on the screen without saving it and move to the first key field. A program should have a line labelled ABANDON: and have statements at line 8470 similar to this:

```
8470 READ (FILE[1],ERR=8475); REM "Remove Lock"
8475 GOTO ABANDON
```

The READ statement at line 8470 ensures that no record is left locked after the **Abandon** command. **Abandon** should clear the data that is displayed, go to the key field, and start the screen over in record maintenance tasks. In addition, it can exit back one level in subwindows where discarding and starting over is meaningless.

Command	Convention
I - Information	The GENINPUT.PUB program traps Information commands and passes them back to the calling program. The GENINPUT routine merged into that program performs a GOSUB INFO. The standard GENINPUT routine contains a generic "do-nothing" INFO section. If 6200 INFO: has been replaced with a functional routine, it will CALL GENINFO.PUB. That public program prompts for a selection from a menu of predefined auxiliary information display tasks. The menu to use is found in the xxMN file with a record key equal to xxINFO.
B - Drill	The GENINPUT.PUB program traps the Shift-F3 command and passes it back to the calling program. The GENINPUT routine merged into that program performs a GOSUB DRILL. The standard GENINPUT routine contains a generic "do-nothing" INFO section. If 6300 DRILL: has been replaced with a functional routine, it will be executed.
C, G, K, N, Q, Y, - Programmer-Defined Commands	The GENINPUT.PUB program can also trap these other commands (on Windows machines in text mode these commands are returned by the Shift+F3 through Shift+F9 keys). Programs using these commands must contain code to trap these commands and to perform the intended task. Some of these commands are used in certain OSAS programs.

Single-Character Commands

GENINPUT has the ability to process additional commands. Set X3=9, X4=1, and X\$=" ". GENINPUT.PUB will not print the default data at entry or exit of the public program and will allow for single-character input that will be returned in X\$. Y\$ will be "E", which will make processing continue after the GOSUB GENINPUT entry of data fields. This eliminates the necessity of pressing **Enter** to enter the command.

This design requires that a command entry position be defined on the screen. The best way to implement this when in character mode is to use `X8$="message"` first, `PRINT @(65,22)`, and then clear `X8$` from line 22 with `X8$=" "; GOSUB GENERROR`. While in GUI mode, you would not want to print the `X8$` message because there will be command buttons available, but you will still need an `X3=9` input to trap these events.

The ASCII value of the commands returned in `Y$` from `GENINPUT.PUB` are always inside the range 1–95. Characters above 95 are translated into uppercase characters by using the `CVS()` function. Data returned in `X$` is untouched.

To make the same screen work in Graphical mode, you must add an invisible GRID control to the `.BRC` file that contains the screen on which you are working. Use the control ID 4999. Then in your program where you prompt for the keystroke, set `GUI_ID` to 4999 as well.

Error Messages

`GENERROR` is used to display error messages in OSAS. Many are system-generated; some can be delivered by the program. The most common method is to set `X9$` to the appropriate message and `GOSUB GENERROR`. Upon return from `GENERROR`, the program should allow reentry of the data field that caused the error, as in the following code:

```
1420 LET NUM=X
1430 IF NUM<1 OR NUM>3 THEN LET X9$="Must Enter 1-3"; GOSUB CLEAR_GLOBAL: GOSUB
1430:GENERROR; GOTO FIELD
```

`GENERROR` clears `Y$` for the program. The program should assign the user-entered data to the variable before displaying the error message so that the data can be seen and corrected after the message is acknowledged.

In GUI mode, you must also clear `STBL("DESTID")` to prevent looping problems when forcing certain fields.

Error messages should follow consistent standards. The error message should read like a sentence with the initial letter capitalized (as well as field names and file names). Range error messages should use one of these conventions:

Must enter x or y.	where x and y are the only valid entries
Must enter x–y.	where x and y define a range of valid entries
Must enter x, y or z.	where x, y and z are the only valid entries

Beginning in version 5.00, verification errors use the O7=8 convention (see “GENERROR” on page 3-29).

File I/O

File input and output routines are often the most sensitive portions of code. Follow the guidelines below for proper error handling and multiuser control.

The term *interactive program* is used when the user is expected to respond to a condition in a short period of time. The term *noninteractive program* is used when the function proceeds to its conclusion without user involvement. Noninteractive programs need to react to some situations and errors differently or without user intervention.

The actual channel number used for each file is stored in the array variable `FILE[ ]`, which contains 100 elements. It is created with `DIM FILE[STBL(“CHANNELS”)]` by the menu program. The global variable `CHANNELS` stores the maximum number of elements available in this array, and is set within `RMSETUP`. The value assigned to each file in `F0` is used as the element number of the array. The actual value inside that element is the real channel number. This was created by the `UNT` function to provide the next available channel number. It was used by `GENOPEN` when that file was opened. `GENOPEN` then put that channel number into the appropriate array element for that file.

File access must use the `FILE[ ]` array and the appropriate element number.

Opening Files

The GENOPEN routine opens files. File opens are generally performed in lines 100–199. Here's an example of opening a file with GENOPEN:

```
0100 LET F0=2, F$="ARCU"+STBL("CID"); GOSUB GENOPEN
```

F0 contains the file reference number by which the file will be accessed with the standard file array. F\$ contains the file name to open.

Closing Files

The CLOSE statement should not to be used in any application program, except

- when the file is being manipulated for a special purpose (some Resource Manager programs and the inquiry window functions do this)
- when a program must open particular files to perform initialization
- if the current task needs more than 100 files open
- as a prelude to erasing scratch files

In general, data files should not be closed. When the program is completed and the application menu is executed, a BEGIN is executed, which closes all files. This structure guarantees that later programs in a chain will also have access to the files. If you must close a file, its array element in FILE[x] must be set to zero. The preferred method of closing data files is to use the GENOPEN routine. The array element is set to zero for you when you use the GENOPEN routine to close the file.

Reading Records

The READ and READ RECORD statements are used anytime the program needs to get information out of a data file. This information cannot be changed. This task may not write back to that data file. If the information must be changed, use the EXTRACT or EXTRACT RECORD statements to get the information.

Here is an example of a READ statement:

```
1100 READ (FILE[3],KEY=COMPID$,DOM=1200) IOL=OSCOMP
```

Here is an example of a READ RECORD statement:

```
1100 READ RECORD (FILE[3],KEY=ITEM$,DOM=1200) INVE$
```

The READ RECORD and EXTRACT RECORD statements are used to read data from a file into a string template.

Use the following conventions to construct the code for the statement:

- Use the file array variable FILE[x] for the channel number.
- Define the END= clause if needed and the KEY= clause is not used (for example, when accessing the file sequentially).
- Use the ERR= statement if special processing other than SETERR GENERROR/GENLOCK is needed.
- Use the IOL= clause with a READ statement, using the appropriate line label for every read from a predefined file. If the program requires only a partial read, you can replace the IOL= clause with hard-coded variables.
- Define the DOM= clause as needed (accessing the file by key).
- Use the DIR= clause to control the movement of the key pointer. DIR=0 on a read leaves the pointer at the record read, negative values move the pointer backwards from the record read, and positive values move the pointer forward.

Writing Records

The WRITE and WRITE RECORD statements are used when the program must put a new or changed record into a data file. A record should be read and locked with the EXTRACT or EXTRACT RECORD statement if it exists before the record is written.

Here is an example of a WRITE statement:

```
1100 WRITE (FILE[3]) IOL=OSCOMP
```

Here is an example of WRITE RECORD logic:

```
1100 LET INUP$=FIELD(INUP$)
1110 WRITE RECORD (FILE[3]) INUP$
```

The assignment on line 1100 strips the trailing nulls from the string template before the template is written to the file. Failure to apply the FIELD() function to the string may result in an error 1. (See section 9 in this chapter for more information about string templates.)

Use the following conventions to construct the code for this statement:

- Use the file array variable FILE[x] for the channel number.
- Use the IOL= clause for every write from a file. Do not write out a record after a partial read.
- Use the KEY= clause only for direct files and Mkeyed files defined with a single key. The IND= clause is used for indexed files.
- The DOM= clause is used only when trying to reserve a record, as in order number control schemes.
- Precede WRITE RECORD statements with the FIELD() function to strip trailing nulls if the string template contains variable-length fields.

Precision setting is required only when you want to force internal stored data to have a specific decimal size. The internal value is not rounded off when reading or writing to a file. Setting PRECISION before file I/O is not required or recommended. Use PRECISION only before a calculation operation where the numeric output needs to be preset.

Multuser Issues

Locking Files

If a task is going to write large quantities of data to a file, you might want to lock the file. If no one should interfere with the write process (such as during post operations), you must lock the file. Individual record writing does not require that a file be locked.

To lock a file, send the F0 file number to the GENOPEN routine as a negative value. If F0 is a negative number, the GENOPEN routine will LOCK the file after it is opened.

The UNLOCK statement is not used in application programs. Data files that are locked upon opening are left locked throughout the program's execution. The lock is released when the data file is closed by the BEGIN in the menu program.

File locks are constraining to users. Do not use them if you can use a better method.

Locking Records

You must use the EXTRACT or EXTRACT RECORD statements when the program reserves a single record for itself to modify. You can also use the EXTRACT statement for multiple record types related to the same main record. This task requires some other manipulations.

Any file operation performed by the same user that EXTRACTed a record on the same channel will unlock that record. Some files contain multiple record types. This usually requires a series of reads to access all the information for a single set of data. If you must access more than one record in the same file, be careful not to lose your record lock.

Below are some strategies for locking multiple records:

Plan 1

EXTRACT the main record on one channel.
OPEN the same file on another channel number.

Perform the balance of the file operations with the secondary channel number.

All programs in all applications must use the same locking scheme for each particular file and record type

Plan 2

READ record type 1.
READ record type 2.
EXTRACT record type 1.

This strategy leaves a small window of vulnerability between the first READ and the EXTRACT, because another user can get in and read the same record for another task and then find that the data has been changed. To make sure that this scheme works and to keep vulnerability to a minimum, put the READs and the EXTRACT tight together in the code.

Tasks that need to access the same record combination must use the same order of access. This means that if two users are in the same function in the same record, one will be denied access, because one will have the EXTRACT first.

Plan 3

Create a temporary record somewhere and lock it.
Perform all file processing.
Remove that temporary record when done.

All programs in all applications must follow the same scheme for each particular file and record type.

Numeric Display and Overflow Checking

Beginning in version 5.00, users can specify the number of decimal places in various types of numeric variables throughout the system. Follow the guidelines below for numeric mask assignment, proper numeric display with numeric overflow (error 43) checking, and precision handling.

Assigning Numeric Masks

Assign your numeric masks at the top of each program chain between lines 300 and 898. The common convention is to use variable names beginning with *M*. In older programs, masks are assigned to variables M0\$, M1\$, and so on. In newer programs, the names are more meaningful, such as MDOLL\$, MQTY\$, and so on.

The setup program assigns to global strings the numeric masks and numeric precision for each type of numeric, based on the user-defined numerics set up in the Company Information function. Here are the global strings used:

Numeric Types	Mask Variable	Precision Variable
Currency/Dollars	+DOLL	-DOLL
Quantities	+IQTY	-IQTY
Unit Cost	+INUC	-INUC
Unit Price	+INUP	-INUP
Credit Limits	+CRED	-CRED
Hourly Rates	+RATE	-RATE
Hours	+HOUR	-HOUR
GL Dollars	+GDOL	-GDOL
GL Dollars w/ Commas	+GCOM	-GCOM

Follow this convention for assigning numeric masks:

```
0300 LET MDOLL$="-"+STBL("+DOLL"); REM "Mask w/ leading minus for screen display
0300 LET MDOLL$=STBL("+DOLL")+ "-"; REM "Mask w/ trailing minus for reports
```

Printing Numbers on Screens and Reports

Once you have established the variables containing the numeric masks, you can use the mask to print a number on the screen or in a report using the FNNUM\$() or FNGUINUM\$() function, which is defined with the GENNUM mergeable routine. For example, to display the variable A5 using the MDOLL\$ mask, use this line:

```
7540 PRINT @(15,4),FNNUM$(A5,MDOLL$)
```

The FNNUM\$() function returns a string containing the result of masking the numeric variable (A5) into the specified mask (MDOLL\$). If the numeric value exceeds the integer mask size, the string will return asterisks (“*”). This prevents mask overflow errors in screen display and report printing.

For GUI screens, the line of code would read

```
7540 PRINT (OSGUI.CHAN) 'TITLE' (4010,FNGUINUM$(A5,MDOLL$,4010))
```

You must pass the control ID into the function. The function then resets the mask for the control.

Precision Handling

Maintain the precision of numeric variables, using either the precision method or the rounding method. In either case it is good practice to determine each precision value and the maximum precision at the beginning of the program.

For example, if your program uses variables with currency, unit cost and unit price precision, set DOPR, UCPR and UPPR to their respective precision values, and MXPR to the maximum of those three values, as follows:

```
0100 LET DOPR=NUM(STBL("-DOLL"))
0102 LET UCPR=NUM(STBL("-INUC"))
0104 LET UPPR=NUM(STBL("-INUP"))
0106 LET MXPR=MAX(DOPR,UCPR,UPPR)
```

The rest of your program should use one of the methods outlined below.

NOTE: The rounding method is the preferred method of handling precision, in that it is always obvious which precision is being used in any calculation.

Precision Method

Set the appropriate PRECISION before mathematical calculations containing multiplication or division, or before functions that perform these calculations internally:

```
1000 PRECISION DOPR
1010 LET A5=A4/X
1020 LET A6=MOD(A4,2)
```

Set the maximum PRECISION before I/O operations:

```
3000 PRECISION MXPR
3010 WRITE (FILE[3],ERR=GENERROR)IOL=ARINXXX
```

Rounding Method

Set maximum PRECISION at the beginning of each program:

```
0110 PRECISION MXPR
```

Perform a ROUND() function on any mathematical calculations containing multiplication or division, or on functions that perform these calculations internally:

```
1020 LET A5=ROUND(A4/X,DOPR)
1030 LET A6=ROUND(MOD(A4,2),DOPR)
```

There is no need to set PRECISION before file writes, because you set the PRECISION to the maximum at the beginning of the program.

Environment

Variables

OSAS software was originally written on a Business BASIC II interpreter, which limited variable names to two characters. Each variable consisted of a letter and a number. The current interpreter allows much longer variable names. Make variable names as meaningful as possible without making maintenance too difficult.

To prevent conflict with other applications, study the other applications' data. This is especially important when you define a new file which is accessed in a program which accesses other data files.

The current interpreter allows for more flexible creation of matrices. String arrays and numeric matrices with variable starting and ending bounds are allowed. The example below shows a matrix with three elements, 1, 2 and 3, instead of the standard four elements, 0–3.

```
0100 DIM A[1:3]
```

Always use this form for arrays. This standard also allows an IOLIST to use the [ALL] option. It makes IOLISTS much smaller and easier to define and use.

Common Variables

Many common variables are defined and used throughout the system. The most commonly used variables are described below. There are three categories of variables: variables set before a program is run, global variables set before a program is run, and variables used in a program.

Below is a list of variables initialized by the menu program before a program is RUN (or CALLED).

Variable	Description
C0\$, C1\$	Current company ID. These variables exist for compatibility with older versions of OSAS programs. Use STBL("CID") for the company ID.
C2\$	Current company name. C2\$ exists for the same reason as C0\$ and C1\$. STBL("CNAME") is the new standard variable for the company name.
FILE[x]	Array of open file channels. The menu program DIMensions this array with zeros. When DIMensioned, x is set to the numeric value stored in STBL("CHANNELS").
GUI	Set to 1 if the user has chosen to run in graphical mode, or to 0 if the user has chosen to run in text mode.
O1\$	Program name to RUN upon completion of the current function. The value is normally "RMMENU." STBL("MENUPROG") also contains the appropriate program name.
O2\$	Option switch settings. The length of O2\$ is 50. Each position represents one option switch. An option switch is ON if O2\$(n,1)="1" and OFF if O2\$(n,1)=" ".
OT\$	Menu title selected. In each program, set TITLE\$=OT\$ before displaying the screen box.
T\$	Table file name. The value is normally the 2-character application code plus "TB". The table name passed to a program in T\$ is the table that relates to the menu file being used. If you run an AP program from a PO menu, T\$ will contain "POTB". Read all tables from the active application's table file.

Variable	Description
X0\$	The program parameter. The menu program sets X0\$ to the current program's parameter entry in the menu record.
OSGUI\$	A template that stores information about GUI screens that are currently being used.

Here is a list of commonly used variables. The menu program does not initialize them before running a program, but program and general routines set and use them.

Variable	Description
F	Device number used in PRINT statements. Use the UNT function to assign the value of F. Reports must print to (F).
F0	An array index associated with each file for the FILE[] array.
F\$	File name for a file operation. This variable is used by both the general file open routine and the general error routine.
Mx\$	Numeric mask variables. Use mask variables for consistent display and entry of numbers.
Y\$	Last command entered by the user.
O7	Used with GOSUB GENERERROR to generate standard messages.
X8\$	Used to display local messages and allow the cursor to return.
X9\$	Used to display messages and force a response.
TITLE\$	Holds the screen title. For consistency, set TITLE\$=OT\$ (see menu variables above).

Variable	Description
X2	Used to tell the help system on which of the possible screens the current field resides. Also used to allow multiple identical fields to access the same help screen. To do so, assign a six-digit number xxyyzz to X2 before calling GENINPUT to identify the help screen to use, where xx is the X0 value, yy is the X1 value, and zz is the X2 value.
X0	Data entry variable.
X1	Data entry variable.
X3	Data entry variable.
X4	Data entry variable.
X9	Data entry variable.
X1\$	Data entry variable.
X\$, X	Data entry variable.
WINDID\$	Window setup variable.
SELVAL\$	Window setup variable.
GUI_ID	Data entry variable.
L0	Line counter.
P0	Page counter.
P9	Interprogram status control.
PROGID\$	File maintenance availability flag.
YDISABLE\$	Command control.
DEVICE\$	Output device set by GENOUT.PUB.

Variable	Description
INFO	INFO=0 means clear Info or Detl status flags from status bar. INFO=1 means display "Info" on the status bar, allow and trap keystroke, and return "I" to calling program in Y\$. INFO=2 means display "Detl" on the status bar.
O1\$	Menu program name. Programs return to the menu by executing the line RUN O1\$.
FLD_HLP\$	Help field ID. Set FLD_HLP\$ if a field uses a common help screen for related fields in an application to cut down on the help file size by eliminating duplicate help IDs.
PG_LEN	Report page length. Set PG_LEN in report programs. PG_LEN is generally 56 for file or printer output and 21 for screen output.

OSAS makes use of global variables for data that applications or public programs frequently need. You access global variables with the STBL function—for example, STBL("CID").

String	Description
"APPS"	All applications currently installed on the OSAS system.
"BANK"	Default bank account ID. This variable carries the default bank account ID stored in the company information record.
"BANNER"	Banner page flag. The system sets this variable according to the workstation defaults. The GENOUT.PUB program checks the flag when the user selects output to a printer or file, and prints a banner page if the value is ON.

String	Description
“BASEDIRPATH”	Base OSAS directory. This variable contains the base directory for the OSAS programs (the directory where the progXX directories reside). For example, the default directory is “C:/osas/” on a Windows server.
“BBJFORMS”	Specified if a BBJ print form is to obtain the choice of printers from the client or server. “0” = use client printers, “1” = use server printers.
“BELL”	Bell flag. This variable determines if the terminal should beep when an error message or verification prompt appears. The system sets this value according to the workstation defaults, which can be toggled between null (off) and 'RB' (on) by pressing Ctrl-G in text mode, or by using the pull-down Modes menu in GUI mode.
“BRKT”	Bracketed field flag. The setup program sets this variable to "Y" if the terminal does not support reverse video for field input, or "N" if the terminal does support it. If it is set to “Y,” GENINPUT surrounds the field with square brackets ([]).
“CAPPS”	Applications installed and active for the current company.
“CHANNELS”	Number of FILE[] array elements. Currently this variable is set to 100. It is used by the menu program to DIMension the FILE[] array.
“CID”	This variable contains the current company ID.
“CLIENT_LAUNCH”	Contains the default command used by the workstation to start up documents and other files by the workstation. This is generally used when no specific file type is specified in File Types, or the start-up program for the file cannot be found.

String	Description
"CLIENT_OS"	Contains the actual operating system name that the OSAS client is running on.
"CNAME"	This variable contains the current company name.
"CTS"	Print compressed to screen. The system sets this variable ON if the terminal supports compressed report printing to the terminal, or OFF if the terminal does not support it.
"DATA1"	The first path on which live data files are stored.
"DATA2"	The second path on which live data files are stored.
"DATA3"	The third path on which live data files are stored.
"DATE"	System date. This variable replaces the DAY function. The menu initializes this value to the operating system's date formatted according to STBL("DATEMASK"). Resource Manager allows the user to edit it. Always use this variable for the system date. Do not use DATE(0), because the operating system date returned by that BB ^x function may be different from the OSAS system date.
"DATEMASK"	This variable contains the date mask for displaying dates. The setup program sets the value from the user-selected date format for the current company. Supported formats are "%Mz/%Dz/%Y" (U.S.) and "%Dz/%Mz/%Y" (European). Both formats generate a 4-digit year. You can force a 2-digit year by adding "+" to the string. For example, the syntax to generate a date with a 2-digit year is DATE(X:STBL("DATEMASK")+"+"), where X is the Julian value of the desired date.
"DESTID"	Control ID of a GUI control that the user clicked so control can jump to that entry field.

String	Description
“DOCUMENTDIR”	This is the directory where the OSAS document files reside relative to each workstation. Document files include the OSAS user’s guides, the productivity reports, and any other documents shared by multiple OSAS users.
“DOCUMENT_SERVER”	This is the directory where OSAS documents are stored relative to the server where they are located. For example, if the files are located on the server’s C drive, the path might be “C:/osas/Document”. The directory path relative to the workstation is stored in the “DOCUMENTDIR” global variable.
“EMAIL”	This is the directory where the workstation’s email client is stored.
“EMAILDIR”	This is the directory where the OSAS e-mail files reside relative to each workstation. E-mail files are limited to the the BBj PDF forms generated by OSAS for forms such as invoices, statements, purchase orders, and vouchers.
“EXIT”	Exit from menu flag. The system sets this variable ON to allow the user to exit from OSAS via the Exit command, or OFF if not. The system sets this value according to the workstation defaults. If the value is OFF, the system requires a program to RELEASE to the operating system when chosen from the menu.
“GLCURRYR”	The GL fiscal year with which the user is working. If STBL(“GLYEAR”)=STBL(“GLCURRYR”), the user is processing in the current GL fiscal year; if not, the user is processing in a prior year.
“GLMASK”	Company GL account mask. GENINPUT.PUB uses this variable to format account number type entries.
“GLYEAR”	The current fiscal year for GL processing.

String	Description
“GUI”	Set to YES if the user chose to run in graphical mode, or to NO if the user chose to run in text mode.
“GUIRESPROG”	Contains a list of the public programs that are ADDR’ed when a function is run in graphical mode. It is used to set STBL(“RESPROG”) when launching a GUI function from the MDI menu. (See also “TXTRESPROG”).
“GUIWIND”	Number of elements in each array in the OSGUI\$ template (currently set to 50). It is used by the menu program to DIMension the OSGUI\$ template arrays.
“HZSCALE”	Contains the percentage of horizontal scaling factor currently in use when running in graphical mode. 100 is considered 100%, or normal graphical size. The user controls this by using the Shift-F7 key on any menu type.
“INFOGUI”	This variable contains the directory where the graphical resource files are stored. (Graphical resource files carry .BRC extensions under Visual PRO/5 and .ARC extensions under BBj.)
“INSERT”	Insert flag. This variable determines if the user has selected typeover or insert mode for data entry. The system sets the value based on the workstation options, which can be toggled between ON and OFF by pressing the Insert (Ctrl+T) key.
“INSTALLPATH”	This variable contains the directory path to the installation media. It is used during application installation.
“INTTYPE”	Stores the type of interpreter being used to run OSAS, either PRO/5 or BBj.

String	Description
“JULDATE”	System date in Julian format. To use this number, you must use the function <code>NUM(STBL("JULDATE"))</code> to convert the global string to a number.
“LIVE”	Live data access flag. The system sets this variable to "0" to allow the user access to live data, or "1" to allow access to sample data only. The system sets this value based on the workstation defaults.
“MAC”	The system sets this variable to “1” on an Apple Mac client, or to “0” on any other client operating system.
“MAP”	Stores the workstation default for the Map Lookup ID.
“MDIHILIGHT”	The workstation option that determines if the BBj MDI menu is using highlighting on the left-hand menu while a mouse is rolled over the menu choices. Possible values are “ON” and “OFF”.
“MENUAPP”	The application from which the function was selected.
“MENUVER”	The version number of the application from which the function was selected.
“MENUDESC”	Title of the menu function. This global is the same as the variable OT\$, with the exception that OT\$ is padded by a space on both ends of the name so it can be used in painting the title of the screen.
“MGW”	Current channel on which the PRO/5 GUI device (always X0) is open.
“MPARAM”	The menu parameter passed to the running program. This is set in the Resource Manager Menus function.
“MPROG”	The name of the program run from the main menu.

String	Description
“NUMOPTS”	Number of options and interfaces switches available per application. Currently this variable is set to 50.
“ONLINE”	Online help flag. The system sets this variable based on the workstation defaults. If the value is ON, GENINPUT.PUB displays a list of valid function keys at the bottom of the screen during processing.
“OPSYS”	The current running operating system under which OSAS is installed and running.
“OPTIONS”	Application options. This variable contains the options for the current application and company ID. Duplicate of the O2\$ variable.
“OSTPL”	BBx channel on which the data file OSTPL is open.
“OT”	Menu title selected. This is the same data that can be found in the variable OT\$.
“PACURRYR”	The PA Payroll year with which the user is working. If STBL(“PAYEAR”)=STBL(“PACURRYR”), the user is processing in the current PA payroll year; if not, the user is processing in a prior year.
“PGDN”	Page Down verification flag. GENERERROR uses this variable to determine whether to require a second Proceed command. The system sets this value from the workstation defaults, and it can also be changed using the Modes pull-down menu in GUI mode.
“PRINT_FILE”	The path on which the current workstation ID is set up to create any print files.

String	Description
“QUICK”	Quick entry flag. This variable determines if a function should use the associated quick-entry tables to determine field and jump stops during entry. The system sets this value according to the workstation defaults, which is toggled between OFF and ON by pressing Ctrl+F in text mode, or by using the Modes menu in GUI mode.
“RESPROG”	Resident public programs. This variable contains a list of programs that the menu program ADDRs before running a function. Other programs may ADDR their own public programs because the menu program drops all resident programs and ADDRs only those listed in this variable.
“RMPREFIX”	Path in which Resource Manager is installed.
“RWDATAPATH”	This variable contains the directory path to the Report Writer data.
“SAMPLE”	Path in which sample data files are stored.
“SAMPWIND”	Sample data flag. The presence of this variable indicates that the user is using sample data rather than live data.
“SCREENX”	The last X coordinate at which a GUI screen was placed. New GUI forms will start at this position.
“SCREENY”	The last Y coordinate at which a GUI screen was placed. New GUI forms will start at this position.
“SCREENS”	The total number of screens visible at any given time when a report is printed to the screen in text mode.
“SORTSEQ”	Contains a unique three-character sequence number used to create scratch files.

String	Description
“SORT_FILE”	The path in which the current workstation ID is set up to create any sort files. If no directory is set up, sort files are created in the active data or sample data directory.
“SYSFIL”	The path in which system files are located. Any non-data files needed by applications are stored here.
“SYSOPEN”	The file extension used to open up both OSAPPL.xxx and OSINFO.xxx .
“TIMEMASK”	Time mask. This variable contains the mask for displaying times. The setup program sets the value from the user-selected time format for the current company. Supported formats are “%h:%mz %p” (12-hour) and “%Hz:%mz” (24-hour/ military).
“TXTRESPROG”	Contains a list of the public programs that are ADDR’ed when a function is run in character mode. It is used to set STBL(“RESPROG”) when launching a text function from the MDI menu. (See also “GUIRESPROG”.)
“UTILSPATH”	This variable stores the path to the development utilities directory, if specified through the Directories function.
“VERSION”	Current Resource Manager version. This version number is used as a default for accessing file conversion routines and file creation files.
“VERSW”	Verification flag. This variable determines whether to verify the Abandon and Exit keys when the user presses them. The setup program sets the value based on the workstation defaults, which can be toggled between OFF and ON by pressing Ctrl+V in text mode, or by using the pull-down Modes menu in GUI mode.

String	Description
"VTSCALE"	Contains the percentage of vertical scaling factor currently in use when running in graphical mode. 100 is considered 100%, or normal graphical size. The user controls this by using the Shift-F7 key on any menu type.
"WEB"	This is the directory where the workstation's Internet browser is stored.
"WHSE"	Default warehouse ID. This variable carries the default warehouse ID stored in the company information record.
"WT"	Contains the workstation default for unit of measure used when entering weights in OSAS applications.
"xxDIR"	Path in which an application program files are stored (where xx is the application ID).
"xxMODDIR"	Path in which an application modified program files are stored (where xx is the application ID).

The system uses global variables to store text-based window colors. OSAS defines 13 window types and four additional text colors. Four colors are associated with each window type: background, text, border and title. This table shows the global variable string associated with each window and color type.

Window	Background	Text	Border	Title
Func Level 1	"FBKG"	"FTXT"	"FBX"	"FTTL"
Func Level 2	"FBKG2"	"FTXT2"	"FBX2"	"FTTL2"
Func Level 3	"FBKG3"	"FTXT3"	"FBX3"	"FTTL3"
Menu Level 1	"MBKG1"	"MTXT1"	"MBX1"	"MTTL1"
Menu Level 2	"MBKG2"	"MTXT2"	"MBX2"	"MTTL2"
Menu Level 3	"MBKG3"	"MTXT3"	"MBX3"	"MTTL3"

Window	Background	Text	Border	Title
Information	"SBKG"	"STXT"	"SBX"	"STTL"
Help	"HBKG"	"HTXT"	"HBX"	--
Inquiry	"IBKG"	"ITXT"	"IBX"	--
Error	"EBKG"	"ETXT"	"EBX"	--
Message	"MSBKG"	"MSTXT"	"MSBX"	--
Verify	"VBKG"	"VTXT"	"VBX"	--
Options	"OBKG"	"OTXT"	--	--
Attention-Warning	--	"AWAR"	--	--
Attention-Caution	--	"ACAU"	--	--
Attention-Alert	--	"AALE"	--	--
Attention-Highlight	--	"AHIG"	--	--

The system also stores user-defined numeric masks in global variables. There are nine numeric types to which the user can assign a numeric precision. The following table shows the global strings that contain various default masks and precision settings:

Number type	Mask	Precision
Regular dollar amounts	" +DOLL "	" -DOLL "*"
Quantities	" +IQTY "	" -IQTY "
Unit Prices	" +INUP "	" -INUP "
Unit Costs	" +INUC "	" -INUC "
Hours	" +HOUR "	" -HOUR "

Rates	" +RATE"	" -RATE"
Credit Limits	" +CRED"	" -CRED"
GL dollar amounts	" +GDOL"	" -GDOL" *
GL dollar amounts with commas	" +GCOM"	" -GCOM" *

*Regular dollar amounts and GL dollar amounts always have the same precision.

Text Function Keys

Application programs use the function and edit keys that the setup program defines when you invoke OSAS. The keys are defined using the 'FL' and 'EL' mnemonics. The keys available are **F1–F10**, the four cursor control keys, **Insert**, **Delete**, **PgUp**, **PgDn**, **Home**, and **End**. If a termcap file is used, the system assumes that the settings in the termcap file are correct. If the user's keyboard has no function keys and/or edit keys, you can use the defaults control and escape sequences. Depending on the terminal, the user may be able to remap the keys in Resource Manager.

Function key usage in GUI mode is determined by the shortcuts assigned to commands when the pull-down menus are defined. Function keys in GUI mode have no default keystroke equivalents.

Following is the default standard:

Key	Default	Description
F1	Esc H	Help Function
F2	Esc W	Inquiry Window
F3	Esc D	Delete Record
F4	Esc O	Other Commands (callable programs and routines, etc.)
F5	Esc X	Abandon Data (was Restart)
F6	Esc F	File Maintenance (add/change data in another function)
F7	Esc M	Exit (was Menu)
F8	Esc L	List/Print screen
F9	Esc U	Undo (Edit command)

Key	Default	Description
F10	Ctrl Z	Delete to End of Line (Edit command)
Shift+F1	Esc A	Online Documentation
Shift+F2	Esc I	Information Windows
Shift+F3	Esc B	Used in some programs for program-specific tasks
Shift+F4	Esc C	Customer/Vendor part no. lookup (in Item ID fields)
Shift+F5	Esc G	Item detail lookup (in Item ID fields)
Shift+F6	Esc K	Lot lookup (in Item ID fields)
Shift+F7	Esc N	Serial number item lookup (in Item ID fields)
Shift+F8	Esc Q	Item Description lookup (in Item ID fields)
Shift+F9	Esc R	Copy field contents
Shift+F10	Esc T	Paste field contents
Down	Ctrl J	Down a field, line, or menu selection
Left	Ctrl L	Left in a field
Right	Ctrl R	Right in a field
Delete	Ctrl X	Delete the character at the cursor
Insert	Ctrl T	Toggle Insert ON/OFF (or lookup in an inquiry window)
Home	Ctrl B	Beginning of field, filelist, page group, or menu page
End	Ctrl E	End of field, filelist, page group, or menu page
PgUp	Esc S	Start Over or Up one page of information
PgDn	Esc P	Proceed or Down one page of information

The user can assign a label or descriptor for each key—the system stores the function key labels in the global variable "FLABEL" and the edit key labels in the global variable "ELABEL". The user can select which keys perform which functions. This should have no effect on the functioning of a program. The appropriate key sequences will be passed on to the application program.

The **Tab (Ctrl+I)** key acts as the **Jump** command in text mode (**Ctrl+J** performs the **Jump** command in GUI mode), the **Backspace (Ctrl+H)** key erases the previous character, and the **Enter (Ctrl+M)** key selects or ends an entry. These keys are not user-definable.

Options and Interfaces

There are two types of interfaces between applications. These interfaces are handled by the **OPTxxx** records in the **xxTB** file for each application. Fifty switches are supported in each table. Each application defines its options switches. These switches are not completely consistent between applications, but in general the following switches apply:

Switch Number	Common Usage
1	General Ledger Interface (“=off, “1”=on)
2	Inventory interface
3-5	Application-specific options
6	Job Cost Interface
7-70	Application-specific options

- *Verification interfaces* are used when a task in one application needs to verify data in another application's data files.

The first step is to test the option switch. An option switch must be defined for all cross-application interfaces.

If the interface is on, this application can open the file and read it. The interface switch must be tested before every I/O operation involving the file.

- *Posting interfaces* are used when a task in one application needs to write data to another application's data files.

The first step is to test the option switch. An option switch must be defined for all cross-application interfaces. If the interface is on, this application can open the file, read a record if needed, and/or write to the file. The interface switch must be tested before every I/O operation involving the file.

Writing to another application's files usually occurs in a post function. This usually means LOCKing any files to which the program will write data.

Guidelines

Use these guidelines to create or modify the options and interfaces for an application:

- There must be a switch for all data file interface situations between products.
- There must be an option switch for all history files.
- The Resource Manager program RMACOPT, which uses the options and interfaces defined in the **xxOI** file, asks if the user wants to share option records or to have unique ones for each company installed on the system. This process will automatically create or erase the **OPTxxx** records based on the user's choice.
- To avoid confusion when testing an option switch (with two possible values), use

IF O2\$(4,1)<>"1" THEN ...

rather than

IF O2\$(4,1)=" " THEN ...

Data Files

xxMN - Menus

The **xxMN** file contains the main menu and submenus for an application. It resides in the sysfil directory. Create it with the _define utility program, or at the BASIC prompt as defined in the *Resource Manager File Descriptions* manual. Maintain it through the **Application Menus** function in Resource Manager. To build menus, follow the conventions below.

- Do not use verbs in menu and function titles except to avoid awkward wording or to clarify the meaning. For example:

Use	Not
Department List	Print Department List
Payroll Transactions	Enter Payroll Transactions

However:

Use	Not
Calculate Checks	Check Preparation
Delete Transactions	Transaction Deletion
Post Expense to GL	Expense Posting

- The function title should describe the function fully—for example, **Payroll Information List**.
- Make menu titles as specific as possible.
- Group menu items logically.
- Put function titles on submenus, rather than on the application main menu.

- Do not repeat information inferred from the menu title. For example,:

Use	Not
File Maintenance	File Maintenance
Payroll Information	Payroll Information Maintenance
Departments	Department Maintenance

- List menu items by the frequency with which they are used. For example:

Use	Not
File Maintenance	File Maintenance
Employees	Payroll Information
Departments	Departments

- Use initial capital letters in menu or function titles. Do not capitalize articles, prepositions and conjunctions unless at the beginning or end of a title.
- Define no more than three menu levels (if possible):
 - The OSAS main menu contains applications only.
 - The application menus point to submenus. Place no tasks at this level.
 - The application submenus execute programs to perform tasks. Do not place submenus at this level, if possible.
 - Transaction entry programs should have their own submenu displayed on the header screen in the entry chain. Use this menu to reduce the number of entries on application submenus.

xxDATA.yyy - Data File Descriptions

The **xxDATA.yyy** (where **yyy** is the version number) file contains the descriptions and file creation information for the application's data files. It resides in the system files (sysfil) directory. Create and maintain it through the **Data File Information** function in the Software Development Utilities.

xxHELP - Help Screens

The **xxHELP** file contains the context-sensitive help screens for the application. It resides in the system files (sysfil) directory. Create it as defined in the *Resource Manager File Descriptions* manual at the BASIC prompt, or with the _define utility program. Maintain it through the Help Screens function in Resource Manager, or when you press the **F6 Maintenance** key when viewing field help.

xxWIND - Inquiry Window Definitions

The **xxWIND** file contains the inquiry window definitions for an application. It resides in the system files (sysfil) directory. Create it as defined in the *Resource Manager File Descriptions* manual at the BASIC prompt, or with the _define utility program. Maintain it through the **Inquiry Window Definitions** function in Resource Manager.

Window Record Key Definition Options

Base window IDs must always be 8 characters long. Alternate window IDs must be nine characters long. You can use one of two possible key structures:

- If you invoke the window from an add/change program, make the key unique. It enables users to modify the window records and have the modifications show up for the one occurrence of the window only. For example, the Customer Maintenance Customer ID field should be named ARACC W1. It is the first window offered in the ARACC program.
- If you invoke the window from a pick screen, you can use a standardized window key. Define this key by the main data it displays. For example, ARCUSTID is a good key for a window that displays a generic AR customer ID list at any pick screen CUST ID field. This key structure lets you use a single window record in many programs, and minimizes the size of the **xxWIND** file.

File Name Options

When specifying a file name or channel number, these rules apply:

- If you specify a FILE[ALL] number for the inquiry window, the file currently open on that channel is used for the inquiry window, even if the file names do not match.
- If you do not specify a FILE[ALL] element, GENWIND.PUB finds an unused channel and opens the specified file for the inquiry window on that channel.
- If you follow the file name with an **x**, it looks for the company-specific file.
- If you follow the file name with **.XXX**, it adds the current terminal ID to the end of the file name and looks for that file (used to inquire on scratch files).
- If you follow the file name with **.SYS**, it adds STBL(“SYSOPEN”) to the end of the file name (used to inquire on **OSAPPL.xxx**).

xxOI - Options and Interfaces

The **xxOI** file contains the option and interface questions and option linkages for each application. It resides in the system files (sysfil) directory and is created and accessed by the **Options and Interfaces** function in Resource Manager.

Define each option switch in an application in the file, including the option prompt, the position in the option string, dependent options, applications required for interfaces, and option grouping information.

xxMSG - System Messages

The **xxMSG** file contains optional messages that can be used by GENERERROR.PUB. It resides in the system files (sysfil) directory and is created and accessed by the System Messages function in Resource Manager.

By defining system messages, code can be saved within programs by replacing X8\$ or X9\$ definitions with references to the file using MX8 or MX9. Messages that are used in several places can be references from one central location, thus making your messages easier to maintain and more consistent.

xxTB - Tables

The **xxTB** file contains application-specific tables, one of which is the options and interfaces for the application. You can store other information that fits the columnar structure of this file in the **xxTB** file. It resides in the application data directory. Create it at the BASIC prompt or with the `_define` utility program. Maintain the options tables through the Resource Manager **Options and Interfaces** function. Maintain other tables through the File Maintenance **Tables** function for each application.

Table record names can be 8 characters long. The table record key is the table record name and the company ID and is always padded on the right to 8 characters.

When you need a table record, code the program to first read the table with the current company ID. If the record is not found in the table, change the company ID to a space and try the read again. If the table is not found with this read, the table cannot be found. A table-not-found condition may or may not be an error condition.

The system uses some tables for storage of data between programs. Because these table names are not documented for end users, users do not know that the tables are stored in the table file. These tables need not follow the naming conventions outlined above. They most often use the terminal ID in the record key (for example, the **USERxxxx** record in Report Writer) and may show up in the maintenance key window in the **Tables** function. They are not user-modifiable.

Company Data

Most data files in OPEN SYSTEMS Accounting Software are named consistently using four to seven character names. The first two letters are the application ID. For company-specific data files, the third and fourth letters identify the data file; the fifth, sixth and seventh are the company ID. For data files that are not company-specific, the third and fourth (and in some cases, fifth) letters identify the data file.

WARNING: Data file names must *not* conflict with program names. You must not use the first four characters of a data file name in a program name unless the program name is 8-characters in length or contains an extension (such as .PUB). For example, the **JOBSxxx** file and Billing Summary Report program JOBSR will conflict when company R is used. The Software Development Utilities contains an audit report you can run to detect possible data/program file conflicts.

Adding Data Files to an Existing System

You can easily create data files for an add-on or enhancement in a system in which the base application's data files already exist. Simply merge the new data files' information into the **xxDATA.yyy** file. Then instruct the user to run the **Data File Creation** function in Resource Manager for the application. The user should respond with **NO** to the **Overwrite Existing Data Files** prompt to make Resource Manager create only the data files that do not exist—the new data files.

Modifying Data Files

Keep these things in mind when you consider modifying existing data files:

- There are some reserved fields in existing data files (see the application's *File Descriptions* manual to find out which ones they are).
- You can redefine existing fields to carry more data or to accept more values.
- You should not add new record types to existing files. Adding multiple record types into one file is a non-normalized standard of storing data, and causes some limitations when using ODBC.
- You can create new data files. A new data file may require new BASIS and GENERAL Report Writer dictionary information, and new sample data. You must also update the **xxDATA.yyy** file to create the new files.

Scratch/Temporary Files

Sort/scratch files are named **SCRT.xxx**, where **xxx** is a unique sequence number stored in the global variable STBL("SORTSEQ").

Names of temporary files must be unique to ensure that the functions that use them work in multiuser and MDI environments.

In versions before 7.0, scratch files were named **SCRTxxx**, where **Txxx** represented the terminal identifier FID(0). For example, a scratch file created on terminal T100 is named SCRT100.

Record Length

OSAS generally uses standard record lengths. Performance improves when you keep records exactly on 64-byte boundaries. If a boundary is close, extend the record length to reach the boundary.

Many files have unused room at the end of each record. Avoid using this room by adding variables to the end of the file's IOLIST when making modifications. If you do, you must change the IOLIST for *every* program in *every* application that reads or writes to the changed file. OSAS applications write entire records; if you do not modify *all* applications that touch the file, the unmodified applications erase whatever information the modified applications put into this space. Instead, make use of the ISV variables at the end of most OSAS data files for your data.

Sample Data

Here are some general guidelines:

- *Applications and enhanced applications always include sample data.* Sample data is a valuable tool for selling software and for training users.
- Enhancement kits may or may not include sample data, depending on the nature of the enhancement.
- *The sample data should be reasonably complete.* Include enough information to demonstrate significant application features and to generate adequate reports. Include both aged, posted data and current, unposted transactions.
- The sample data must follow Generally Accepted Accounting Principles.
- *The sample data should be correct.* Double-check your numbers to make sure that they make sense. Be especially wary of unreal numbers, negative numbers in fields that can't possibly go negative, and misspelled words.

- *The dates used in sample data must be for the fourth quarter 2002 and all of 2003.* These years are updated in each major OSAS release. You should update your sample data dates to match the OSAS ones.
- *The sample data should be original.* When you generate new sample data, do not use names of real people or real companies or copyrighted names of fictional characters or companies.
- The sample data must be consistent between files and applications.

Standard Open Systems Sample Data

Open Systems uses Builder's Supply (company H) as its sample data company and constructs all sample data around simulated 2003 business.

If your product requires unique sample data, use a different sample data company.

Report Writer

Open Systems no longer supports the old Report Writer application. Any inquiry windows or global inquiry windows must be redefined to avoid using RW Topics.

The BASIS Data Dictionary

The BASIS Data Dictionary is a set of data files that describe the contents of the data files used in OSAS for use with the ODBC driver that is included in the OSAS ODBC Kit. The dictionaries are also used by the Software Development Utilities to print file layouts, created string templates, and so on.

The files carry the file extension .1, and the ODBC driver uses the files named that way. Here is a list of the files that make up the BASIS Data Dictionary:

File Name	Contents
FILE.1	File information
FILMSC.1	Additional file information

File Name	Contents
FILRUL.1	File-based I/O rules
FLDRUL.1	Field-based I/O rules
INDEX.1	File key and index information
LOCAT.1	Field locations in data records
RULE.1	General I/O rules
TYPDEF.1	Field-type definitions
TYPRUL.1	Field-type I/O rules
FIELD.1	Field information
VCOL.1	Field information for data views
VMAIN.1	General information for data views
VTBL.1	File information for data views

Build BASIS Data Dictionaries

Each application must include the BASIS Data Dictionary files for its own data files only. The files must be named with the extension **.XX**, where **XX** is the application ID. You can build and maintain the data dictionaries for an application in three ways:

- use the utilities in the Software Development Utilities
- use the utilities in the ODBC Kit
- use the BASIS DDBuilder utility included in the Developer's Tool Kit.

The application dictionaries (***.XX**) are merged into the master dictionary files (***.1**) during the application installation process using the **RMMERGE.PUB** program. You do not need to add this as a last task in your **xxAPPL.TXT** file as the installation will look for these files, and only merge them if found.

To create the mergeable files, build the definitions for the application. Then use the Software Development Utilities to create a new set of dictionary files with the extension from the application ID. Finally, use the SD Utilities to copy the data dictionary information for the application to the new files.

GENERAL Report Writer Dictionary

The GENERAL Report Writer Dictionaries are a set of files that define data file contents and reports for use with GENERAL Report Writer. These files are created using functions within GENERAL Report Writer, and are then split into application-specific installation files by the GNSPLIT utility. This utility is found on the GENERAL Report Writer media, and will split an application's dictionaries and reports into **GEN6MST.xx**D and **GEN6MST.xx**R files. Then, include these files in the **xxRWDATA.ZIP** files for installation. The installation process will then merge these files into GEN4MST.

Other Files

Several other files are used for other purposes. These files are application specific, and get merged into other files at the time of installation. They include:

- **xxCNVT** - Application File Version Information. This file is merged into OSCNVT, and contains all data files used in each version of the application.
- **xxCNVT.HDR** - Application Version Information. This file is merged into OSCNVT.HDR.
- **OSFD.xx** - Change Fields detail file. This file is merged into OSFD.
- **OSFH.xx** - Change Fields header file. This file is merged into OSFH.
- **xxFRM** - Application Form Types file. This file is merged into OSFRM.
- **xx.TPL** - Application template file. This file is merged into OSTPL.
- **xxTPL.700** - Prior version's application template file. This file is merged into OSTPL.700.
- **xxMNU.MRG** - This text file is found only on applications that merge menu entries into base application menu files.

The installation looks for these files automatically, and if they are found, merges them into the appropriate files. Include any merge files that apply to your application on the media.

Installation

Normal installation creates directories, copies files, and updates the main menu.

The OSAS installation process interprets the existence and the contents of the installation files and performs tasks accordingly. If you define the programs and files properly, your product will install easily and uniformly.

Applications must be installed with the standard OSAS installation procedure. Your application must meet these requirements:

- It must use a unique 2-character application ID to distinguish it from other applications available from Open Systems. If you are unsure about the ID to use, check with Open Systems to find out which 2-character application IDs are in use.
- The **xxAPPL.TXT** file must contain the appropriate application information.
- There must be programs for the functions that make up the application, and they must conform to the standards outlined in this document.
- Your sample data must allow the application to function in demo or training mode without a data installation step. This data must be related and logical.
- There must be BASIS Data Dictionaries defining all fields in all data files for the application. Use the functions on the **Data Dictionary** menu in the Software Development Utilities to create the dictionary.
- There must be an **xx.TPL** file that contains string templates that define all files for the application.
- If the application uses options, you must create an Options and Interfaces (**xxOI**) file that contains the questions and data necessary to prompt for the settings for the options switches.

- If the application requires a conversion to a higher version number or modifies OSAS files, you must create programs, **xxCNVT.yyy** (where **yyy** is the version number), which convert the data for each version. The user executes this program through the **Data File Conversion** function in Resource Manager. New files designated as destinations for converted data are created automatically through the normal Data File Creation process. You must also create or edit the **xxCNVT.HDR** file to include the steps required in a multi-version conversion. The **xxDATA.yyy** file must also be kept accurate.
- Use the functions in the GENERAL Report Writer application to create the GENERAL Report Writer dictionary files for the application.
- Modify the **xxAPPL.TXT** file to include any required steps for merging files or other installation tasks.

Application Reinstallation/Upgrade

Design your products so that they can be reinstalled easily. When a product is easy to reinstall, the process of installing upgrades and enhancements is easy, and the user can recover easily if program files are corrupted.

The reinstallation/upgrade process is similar to the new application installation process except that it does not overwrite live data files of an existing application. This prevents the loss of information that the user has created or modified.

Do not do anything that destroys the user's existing data. This is a paramount rule of installation.

Add-on Application

An add-on application is different from a reinstallation/upgrade in that the **xxAPPL.TXT** information indicates that it is an add-on to an installed base application. An add-on application adds programs and/or data files that are used in conjunction with the existing programs and data files.

Create a new set of menus to access the add-on.

Installable add-on application media must meet the same requirements as application media.

An add-on application must not affect existing live data files.

Enhancement Kits

An enhancement kit is different from a reinstallation/upgrade in that the **xxAPPL.TXT** information indicates that it is an enhancement to a previously installed application.

An enhancement kit modifies existing and/or adds application programs and/or data files. After installation, a flag in the general installed application information file is set to indicate an enhanced application. This flag will allow reinstallation of only the core application, the enhancement, or a qualified “relative” enhancement or add-on application.

The menus for the previously installed application being enhanced are replaced with new menus.

Installable enhancement kit media must meet the same requirements as application media. An enhancement kit will not affect live data files during installation. Use a conversion program to implement changes to existing live data files.

Update Kits

An update kit updates applications that are installed on the system. The installation process searches for all the valid applications to update and copies only the programs and data files for applications found. The update kit contains only programs and data files that need to be updated to a higher version, not full applications. An update kit may contain several applications on the media, although only a few may need to be installed. Update kits must contain the file **UPDATE.TXT** on the root directory of the first volume of media.

Installation Flow

Refer to the Resource Manager technical documentation for details of the installation processes. This is a synopsis of the steps involved:

- Prompts for the location of the application to install.
- Checks the location for the existence of **xxAPPL.TXT**. If this file does not exist, the system terminates installation.
- If **xxAPPL.TXT** contains commands in its FIRST TASK section, executes the special processing specified.
- Compares the information in **xxAPPL.TXT** to the general installed application information to determine whether this is a reinstallation, an upgrade or add-on, an enhancement, or a new installation.
- On new installations, prompts for the application's destination directory.
- Loads the contents of the files section of the **xxAPPL.TXT** file.
- Prompts for which set(s) of files to install: programs, system files, sample data, data dictionaries, online documentation, documents, or graphical resource files.
- Copies the files to their appropriate destinations.
- Unzips zipped files on BBJ systems, and if installing from tar media on non-Windows PRO/5 systems.
- Creates an unzip batch file and adds zipped file names to it (on Windows-based PRO/5 systems only).
- Finishes copying the application files and then merges the application information into the general installed application information.
- If the verification data file for a company exists, adds the application to the main menu (both live and sample) for that company.

- If the **xxAPPL.TXT** file contains commands in its LAST TASK section, executes the commands therein.
- Returns the user to the menu on BBj systems, or if tar media is being installed on non-Windows PRO/5 systems, or if there were no zipped files.
- If the Windows installation media contains zipped files on a PRO/5-based system, the system launches a command window to the operating system and executes the unzip batch file. The menu then reappears.

Directories on Distribution Media

Distribution media uses subdirectories to direct the installation process where to copy the files. You will typically use several subdirectories for application media.

The directories in the following descriptions are relative to the path where the installation files for an application reside on the distribution media. If you use diskettes, the root directory refers to the actual root directory on the diskette. If you use a CD-ROM for distribution, the root directory refers to the subdirectory on the CD-ROM that corresponds to that application.

/ (root)

The root directory on the first volume of an application's distribution media contains the **xxAPPL.TXT** file. The system copies file names with no directory name in the FILES section of **xxAPPL.TXT** from the root directory of the distribution media to the programs directory of the application being installed.

/sysfil/

Beginning in version 6.00, system files that were formerly stored in the progRM and progxx directories are stored in a directory called **sysfil**. The **sysfil** directory allows the client/server versions of OSAS to address system files using the data server for faster access and use with Windows workstations.

/progxx/

The application programs directory contains all the application programs and application-specific data files. There may be more than one application program directory on the distribution media. For example, General Ledger media has both progGL and progRM, for GL application programs and programs that allow other applications to interface with GL, respectively.

/sample/

The sample data directory contains data files with sample data.

/data/

The data directory typically contains only the application table file, **xxTB**.

/RWdata/

The Report Writer data directory contains mergeable GENERAL Report Writer files, Global Inquiry merge files, and ODBC data dictionaries.

/gui/

The GUI directory stores all graphical resource files (.BRC files under PRO/5, and .ARC files under BBj). These files are accessed only when operating in graphical mode

/Document/

The document directory stores all productivity reports for an application.

/utils/

The utilities directory contains utilities not installed with the rest of the media.

Distribution Media File Descriptions and Comments

Pre-Installation Files

In addition to the application programs and data, include an xxAPPL.TXT file on the distribution media (xx stands for the application ID) that describes the application and its installation. Below is a sample of the POAPPL.TXT file:

```
[Application Information]
ApplId = PO
ApplDesc = Purchase Order
InstDesc = Purchase Order
ApplSDesc = Purchasing
ApplVFile = POORx
AddApp = AP
ApplFlag =
ApplVer = 7.01
ApplSize = 5544

[Dependencies]
ApplDep = AP~

[Install Files]
File001 = POAPPL.TXT
File002 = RWdata/PORWDATA.ZIP
File003 = data/PODATA.ZIP
File004 = gui/PORES.ZIP
File005 = progAP/POPROGAP.ZIP
File006 = document/POPROD.ZIP
File007 = progRM/POPROGRM.ZIP
File008 = sample/POSAMPLE.ZIP
File009 = sysfil/POSYSFIL.ZIP
File010 = document/PODOC.PDF

[First Task]
cmd001 =

[Last Task]
cmd001 = CALL STBL("RMPREFIX")+ "RMAPPL.PUB", "AP", "OFF"

[Label Information]
Label01 = Purchase Order 7.01
Label02 = Open Systems Holdings Corp.
Copyright (c)1982 - 2005
All Rights Reserved.
```

This text file contains a number of sections and tags that describe the application and offer key information about how the application is installed. Section titles are delineated by square brackets ([]), while tags are found at the beginning of each line and are followed by an equal sign and then a developer-assigned value.

Section titles and tags must appear exactly as described below.

[Application Information]

This section contains several tags that describe the application.

Tag	Description and Comments
ApplID	The two-character application ID.
ApplDesc	The name of the application as it will appear on the graphical and text OSAS main menus.
InstDesc	The name of the application as it will appear on the installation menu.
ApplSDesc	The short description of the application as it will appear on the application tabs in the MDI menu.
ApplVFil	The filename for verifying that the application is installed. Use a lowercase x in the filename to indicate that the file is company specific.
AddApp	The two-character ID of the application onto which this application adds functionality or on which it is dependent.
ApplFlag	Left blank on OSAS shipping applications, this field can be used for modifications to prevent reinstallation of the original (unmodified) OSAS application.
ApplVer	This is the version of the application to be installed.
ApplSize	The number of kilobytes of disk space consumed by the installed application, not including data files.

The system uses the values for the application code, enhancement code, and enhancement ID in a manner illustrated in this table:

Tags in <i>xxAPPL.TXT</i>	AppIID	AppIFlag	AddApp
Application Code	xx	xx	yy
Enhancement Code	blank	xx	xx
Enhancement ID	blank	site ID	blank

[Dependencies]

The Dependencies section lists applications that must be installed before this application, if any. The installation process will ensure that if multiple applications are installed at one time, dependent applications are installed after the applications on which they depend.

Tag

AppIDep

Description and Comments

The two-character application IDs of each application that must be installed before this application. Each application ID must be followed by a tilde (~).

[Install Files]

The Install Files section contains a list of the files that will be copied to the server during the installation process. File names without subdirectory information are copied to the application's progXX directory. To install files to other directories on the server, include the subdirectory information in the file name.

Tag

FILE*nnn*

Description and Comments

The name of each file and subdirectory, if necessary, to be copied during the process. *nnn* must be a unique 3-digit number entered in consecutive ascending order beginning with 001.

Enter names in each directory in alphabetical order, beginning with files in the root directory. Enter subdirectories as needed. For example:

FILE001 = File1

FILE002 = File2

FILE003 = data/XXDATA/ZIP

FILE004 = progXX/XXPROGXX.ZIP

File names can refer to zipped files or unzipped files.

Capitalization is critical to proper installation. See **xxREMOVE.TXT Standards** below for more information about capitalization rules.

[First Task]

This section contains BBx commands that must be executed by the installation process before installing the application. A typical use of this capability would be to CALL a program stored on the initial distribution volume that does something out of the ordinary to the installed OSAS as a prerequisite to installation (for example, verifying that a certain version of RM was installed).

Tag	Description and Comments
<i>cmdnnn</i>	Optional preinstallation processing. For example, <code>cmd001 = CALL INFODEV\$+"DO-THIS.1ST"</code> To prevent losing access to the installation process, do not use a RUN here. Note the use of INFODEV\$. It forces access to the installation directory for the program so that it doesn't have to be copied.

[Last Task]

This section contains BBx commands that must be executed by the installation process after installing the application. A typical use of this capability would be to turn off access to another application or to merge files after installation.

Tag	Description and Comments
<i>cmdnnn</i>	Optional postinstallation processing. For example, <code>cmd001 = CALL STBL("RMPREFIX")+"DO-THIS.LAST"</code> To prevent losing access to the installation process, do not use a RUN here.

[Label Informaton]

This section contains the name, version and copyright information for the application as it will appear on the application information inquiry screen.

Tag	Description and Comments
Label01	The first line of the copyright information, specifying the name of the application and the version, followed by the right-justified copyright years.
Label02	The last line of the copyright information, specifying the copyright holder's name and the right-justified text "All Rights Reserved."

Post-Installation Files

In addition to the Last Task section of the xxAPPL.TXT file, the system uses these files at the end of the installation process.

Name	Type	Description and Comments
xxREMOVE.TXT	TEXT	Noninstallation processing replaces xxFILES.TXT only for removing files/ The file contains a list of all the files installed for the application. The system uses it when the user chooses to remove an application.
UPDATE.TXT	TEXT	Update kits only • always on first volume root directory • used for list of applications on media • not copied The file contains a list of application IDs on the media in the order in which the system must install them. Separate each ID by one space. The file should use only one line of text, such as: AP AR BR FA GL JO PA PO RM RW SO

To create an installable update kit, follow these steps:

- Copy the **xxAPPL.TXT** file for each application to the root directory on the media.
- Copy each application's specific files into their proper subdirectories.
- Create a file on the root directory named **UPDATE.TXT**, as described before.
- If an application spans more than one volume, the file **xxAPPL.TXT** must be on the root directory of all additional volumes containing the application.

Application Setup Files

The system uses these files after the installation process, during setup:

Name	Type	Description and Comments
xxDATA.yyy	MKeyed	data file information file • used to create, rebuild, report on, and remove application data files • copied to application programs directory
xxCNVT.yyy	Program	data file conversion • prompt user for location of old and new files • copied to application programs directory • optional

Zipped Media

When creating zipped media for an installation, follow these guidelines:

1. Create a ZIP file for each subdirectory.
2. Place each ZIP file in the subdirectory where the system will unzip it.
3. If the subdirectory is RWDATA, create these ZIP files and place them in the RWdata subdirectory:
 - **xxRWDATA.ZIP** - all RW files in the RWDATA subdirectory
 - **xxRWGIDF.ZIP** - global inquiry window definition files (optional)
4. Add the ZIP file names to the INSTALLATION FILES section of **xxAPPL.TXT**.

Each ZIP file that you create for a subdirectory should be named **xxPROGxx.ZIP**, where **xx** is the application ID and **PROGxx** is the application subdirectory. For example, to zip the Accounts Receivable program directory, you would name the file ARPROGAR.ZIP.

If you need additional ZIP files, name each additional file **xxPROGyy.ZIP**, where **yy** is a sequence number. For example, name the first zip file ARPROGAR.ZIP, and additional files ARPROG02.ZIP, ARPROG03.ZIP, and so on. Originally this was done for media to expand beyond a single floppy diskette. Since OSAS no longer distributes media thru diskettes, base OSAS media no longer splits out ZIP files between multiple files within the same directory.

ZIP files for data files, sample data files, and system files, if necessary, should be named **xxDATA.ZIP**, **xxSAMPLE.ZIP**, and **xxSYSFIL.ZIP**, respectively.

The ZIP file containing the graphical resource files (.BRC files for Visual PRO/5 or .ARC files for BBj), must be named **xxRES.ZIP**.

Application Files

xxREMOVE.TXT Standards

You must create an **xxREMOVE.TXT** file using the following guidelines to ensure consistency in removing applications for all platforms.

- Enter file names in each directory in alphabetical order. Start with files in the root directory. Then enter subdirectories in alphabetical order. For example:

```
File1
File2
.
.
data/File1
data/File2
.
.
progXX/File1
progXX/File2
.
.
sysfil/File1
sysfil/File2
.
.
RWdata/File1
RWdata/File2
.
.
gui/File1
gui/File2
.
.
sample/File1
sample/File2
```

- Capitalize file names that are BB^x program files or data files.
- Capitalize file names with the extension “TXT.”

- Do not capitalize UNIX script files or UNIX executables.
- Subdirectories must be all lowercase, unless they contain an application ID (the Documents subdirectory is an exception). Capitalize application IDs in the subdirectory name, for example:

```

/DATA becomes /data
/PROGAR becomes /progAR
/RWDATA becomes /RWdata
/SAMPLE becomes /sample
/SYSFIL becomes /sysfil
/DOCUMENT becomes /Document
/GUI becomes /gui
/UTILS becomes /utils

```

System Files

Resource Manager uses these data files:

Name	Type	Description and Comments
xxHELP	MKeyed	help file • used from general input routine for context-sensitive help • copied to system files (sysfil) directory
xxMN	MKeyed	menu file • used by the main menu processor • menu with type, description, function, and parameter • copied to system files (sysfil) directory
xxWIND	MKeyed	inquiry window definitions • used from the general input routine • window with size and field definitions • copied to system files (sysfil) directory

Name	Type	Description and Comments
xxOI	MKeyed	options and interfaces • used by the RM Options and Interfaces function • option description, valid entries and definitions • copied to system files (sysfil) directory
xxDATA.yyy	MKeyed	options and interfaces • used by the RM Data File Maintenance functions • used by Create and Convert Data Files functions • definitions of application data files for each version • copied to system files (sysfil) directory
xxMSG	MKeyed	system messages • used by RM System Messages function • used by GENERERROR.PUB for messages • copied to system files (sysfil) directory

Using String Templates

Definition

Use string templates as another way to store data. Simply define a template that will define the data elements stored in a string. Here are some samples of string templates:

```
"CUST:C(10),NAME:C(30),BALANCE:N(14)"
```

This syntax means that the variables CUST, NAME, and BALANCE are stored in the string as follows:

Variable	Description
CUST	alphanumeric field, with a length of 10
NAME	alphanumeric field, with a length of 30
BALANCE	numeric field with a length of 14

```
"CUST:C(10*),NAME:C(30*),BALANCE:N(14*)"
```

This syntax means that the variables CUST, NAME, and BALANCE are stored in the string as follows:

Variable	Description
CUST	alphanumeric field, with a length of 10, followed by a field terminator
NAME	alphanumeric field, with a length of 30, followed by a field terminator
BALANCE	numeric field with a length of 14 followed by a field terminator

This syntax does not mean that the CUST variable is forced to a length of 10; if a field is field-terminated, it can be any length.

To assign this template to a string, use one of these standards:

```
0010 DIM ARCU$: "CUST:C(10*),NAME:C(30*),BALANCE:N(14*)"
```

or

```
0010 LET B$="CUST:C(10*),NAME:C(30*),BALANCE:N(14*)"
0020 DIM ARCU$:B$
```

This assigns the template to ARCU\$. You can then reference the fields inside ARCU\$ as follows:

```
0030 LET CUSTOMER_ID$=ARCU.CUST$; PRINT ARCU.NAME$;
0040 LET CUSTOMER_BALANCE=ARCU.BALANCE
```

A rule of thumb for string templates is that if the contents of a variable is to be handled more than twice, assign the variable to a local variable, manipulate the local variable as needed, and reassign the string template.

File Access

When you read and write to a file, use the READ RECORD and WRITE RECORD statements.

```
0010 LET KEY$="ACE001"
0020 READ RECORD(FILE[1],KEY=KEY$,DOM=0030,ERR=GENERROR)ARCU$; GOTO 0040
0030 LET O7=8; GOSUB GENERROR; GOTO READ_NEXT
0040 LET ARCU.BALANCE=NEW_BALANCE
0050 LET ARCU$=FIELD(ARCU$)
0060 WRITE RECORD(FILE[1])ARCU$
```

Here is an explanation of the code:

0020 reads the record from FILE[1] into ARCU\$ template

0040 sets up a new balance in ARCU.BALANCE

0050 clears extra data from the end of the ARCU\$ template

0060 writes the record

Line 50 is required before you write any record into OSAS files. As the READ RECORD statement reads the record into the ARCU\$ template, BBx pads the end of the templates with nulls (\$00\$) up to the length of the record from the data file. In this case, the template would be padded with enough nulls to bring the template up to 1024 bytes (the record length of the file). The FIELD clause on a template will clear the trailing nulls from the end of the template.

Field Types

```
"CUST:C(10),NAME:C(30),BALANCE:N(14)"
```

The bold letters above define the type of field in that portion of the string. You have several options:

Variable Types	Description
C	character string
N	numeric string
I	signed binary integer
U	unsigned binary integer
F	IEEE binary floating-point value (8 bytes)
D	BCD floating-point value (8 bytes)
B	BB ^x business floating-point value (8 bytes)
X	local float (C) value (4 bytes)
Y	local float (C) value (8 bytes)

Clearing Templates

To clear a template (similar to reDIMensioning an array), use the following syntax:

```
0070 DIM ARCU$:FATTR(ARCUS)
```

This statement clears ARCU\$ but keeps the template definition in ARCU\$ intact.

Other Functions

A field in a template can be accessed in two ways. You can access the data directly, using

```
0030 LET CUST_ID$=ARCU.CUST$, CUST_NAME$=ARCU.NAME$, CUST_BALANCE=ARCU.BALANCE
```

The FIELD() and NFIELD() functions can also be used to access data in a string template as follows:

```
1000 LET CUSTOMER_ID$=FIELD(ARCU$, "NAME")
```

or

```
1000 LET A$="NAME", CUSTOMER_ID$=FIELD(ARCU$, A$)
```

With this format, you can determine field values at run time, and by changing a variable that stores the actual template field name, you can write generic programs that retrieve a field from a template.

For numeric fields, use this syntax:

```
1000 LET CUSTOMER_BALANCE=NFIELD(ARCU$, "BALANCE")
```

or

```
1000 LET A$="BALANCE", CUSTOMER_BALANCE=NFIELD(ARCU$, A$)
```

If you are referencing a numeric in the string template, you must use the NFIELD function.

SELECT Verb

One of the most useful things about using string templates in your applications is the availability of the SELECT verb for reading records from a data file. The SELECT verb handles record selection internally, and returns only records that match the criteria you specify.

Open Systems has implemented SELECT in the Inventory and Bill of Materials/Kitting applications for creating scratch files, for record searches, and for promotional price record selection.

In its simplest form, SELECT returns all records from a data file:

```
0010 LET CHAN=UNT,F$="INVE"+STBL("CID")
0020 DIM INVE$:FATTR(INVE$)
0030 SELECT (CHAN) INVE$ FROM F$
```

Line 0030 uses SELECT to request all the records from the file specified in F\$ (INVExxx). After the SELECT you can simply perform READ RECORDs to access the information returned by SELECT from the file channel specified by CHAN:

```
0040 READ RECORD(CHAN,END=REPORT_DONE) INVE$
0050 PRINT ....
0060 GOTO 0040
```

When an end of file is encountered, all the records returned by the SELECT verb have been read.

SELECT becomes more powerful when you use the WHERE clause to limit the scope of the records returned. The following statement returns only the items with a status of Discontinued or Obsolete:

```
0030 SELECT (CHAN) INVE$ FROM F$ WHERE INVE.STAT$="D" OR INVE.STAT$="O"
```

The SORTBY clause can also be added to the SELECT verb to determine the order the records are returned. The following statement returns obsolete and discontinued items sorted by the status and then by the item ID:

```
0030 SELECT (CHAN) INVE$ FROM F$ WHERE INVE.STAT$="D" OR INVE.STAT$="O" SORTBY
0030: INVE.STAT$+INVE.ITEM$
```


Data File Conversion

To provide for multiple-version conversions of data files, the naming convention of the conversion programs and the **xxDATA** files changed in version 5.10. The new convention is **xxCNVT.yyy** and **xxDATA.yyy** (where **xx** is the application ID and **yyy** is the destination version number). For example, APCNVT.450 is the conversion program that converts AP from 4.00 to 4.50.

xxCNVT File

To determine which conversion program to start with, Resource Manager uses the xxCNVT data file for the application. The **xxCNVT** file stores the following data for each data file in each version:

Description	Variable Name	Len
Application ID	AP\$	2
Version	VER\$	5
File name	OSFIL\$	12
Message	MSG\$	1

To create the file, use the **Create xxCNVT File** function on the SD Utilities **System File Maintenance** menu. Enter one record for each file in each version. For company-specific files, use a lowercase **x** (when verifying files, the **x** is replaced by the company ID). For example, if the application starts at version 3.20, you need a record for each conversion step.

AP	3.20	APCHx
AP	4.00	APCHx
AP	4.10	APCHx
AP	4.50	APCHx
AP	5.00	APCHx
AP	5.10	APCHx
AP	6.1x	APCHx
AP	6.5x	APCHx

AP	7.0x	APCHx
AP	7.5x	APCHx

The **xxCNVT** file is merged into the OSCNVT file. OSCNVT keeps all the verification information for all applications that are installed on the system.

xxDATA.yyy File

You need an **xxDATA.yyy** file for each version to which you convert. For example,

APDATA.410
APDATA.450
APDATA.500
APDATA.510
APDATA.610
APDATA.650
APDATA.700
APDATA.750

You should be able to take the old **xxDATA** files from previous applications and just rename them to the proper format. Notice that because there were no changes to data files with version 6.0, there is not an APDATA.600.

xxCNVT.yyy Program

For help in upgrading your conversions, use the skeleton program named SDSKCNVT in the progSD directory of the Software Development Utilities. There are two ways to use this program to upgrade your existing conversion programs to work in the new chained format. Method 1 merges pieces from the skeleton program into your existing conversion program. Method 2 merges the main conversion lines from your existing program into the skeleton program. Either method produces the same result.

Method 1

1. Rename your current conversion program **xxCNVT.yyy**.
2. Change line 1 to read SETERR ERRORTRAP.

3. Remove the verification and error log logic from your program. Resource Manager will now perform the verify and print the error log automatically. The verification and error log logic is traditionally located at lines 89–110 and lines 1000–1999.
4. Check that all your main conversion lines are between lines 2000 and 3999. If your main lines go beyond this, renumber these lines. You can renumber these lines into the 1000 block because it is no longer needed for verification, resulting in a workspace from lines 1000–3999.
5. If you need to copy files rather than convert them, write down the file names that will be copied.
6. Check your program for custom code in these line ranges. The merge will overwrite these sections:

Lines 4000-4399	copies files
Lines 4500-4599	new file open logic
Lines 4600-4899	writes out to the error log
Lines 5300-5399	error trapping
Lines 8400-8499	GENERERROR
Lines 9000-9155	GENDATE
Lines 9990-9999	QUIT logic

If you have custom code in these areas, renumber it to an unused area.

7. Use the list above to merge in the pieces from the SDSKCNVT program.
8. Change file opening logic as follows:

old: GOSUB 5200

new: GOSUB 4500
9. Change file error message logic as follows:

old: GOSUB 5695

new: LET LOG=1; GOSUB 4800

10. Change any other messages that have been used on the error log, based on the chart on the next page.

Number	Log Message
1	<filename> does not exist. It could not be converted.
2	<filename> had an invalid date. It was changed to (Blank).
3	<filename> does not exist. It could not be copied.
4	Orders exist in the order file. File was not converted.
5	<filename> at key <keyp> has an invalid address. The address was not converted.
6	<filename> at key <keyp> is an invalid record. The record was not converted.
7	An error occurred while converting the record at key <keyp> in file <filename>. This record was skipped.
8	An error occurred while converting the record <keyp> in file <filename>; the data was converted and conversion continued.
9	Error opening <sourcdir>FAMA+STBL("CID") file. Cannot create credit amount for retired asset record.
10	Cannot find <R0\$(1,14)> in the Asset Master file. Cannot create GL account and accumulated eligible deduction.
11	Error reading <OLDRO\$> in <sourcdir>FAMA+STBL("CID") file. Cannot create credit amount for retired assets.
12	Error loading APRCDx table. Cannot create v4.0 recovery period fields.
13	Error writing new APRCDx table to v4.0 FATB.
14	Error opening the old FATAx file or the v3.20 credit tables.
15	Depr. table ID not translated. Current value remains and must be changed by user.
16	Error opening FAMAx file. Cannot create GL account and accumulated or eligible accumulated.

Number	Log Message
17	The <keyp> subassembly is serialized or lotted and has been changed to an item type.
18	<filename> at key <keyp> has an invalid state code.
99	ERR = <errnum>, the line number was <line>, the program name was <prog\$>, and the key of the data file was <keyp>.

11. Any other messages that are not in the chart can be printed by setting MSG\$ before calling RMCNVTLG.PUB.
12. To copy files, GOSUB 4200 and use the following logic for each of the file:
13. LET F\$="<filename>"+STBL("CID"); GOSUB COPYFILE
14. Change line 9997. For the last program in the chain (the current version),

```
LET PGM1$="RMCONVER"
```

to let the driver know that the application is finished. For all other programs in the chain,

```
LET PGM1$=APPLDIR$+FILENAME$; REM "the next conversion program in the chain"
```

Method 2

1. Rename the skeleton program (SDSKCNVT) to **xxCNVT.yyy**.
2. Check that all your main conversion lines are between lines 2000 and 3999. If your main lines go beyond this, renumber these lines.
3. If you need to copy files rather than convert them, write down the file names that will be copied.
4. Check your program for custom code in these line ranges. The merge will overwrite these sections:

Lines 4000-4299	copies files
Lines 4500-4599	new file open logic
Lines 4600-4899	writes out to the error log

Lines 5300-5399	error trapping
Lines 8400-8499	GENERERROR
Lines 9000-9155	GENDATE
Lines 9990-9999	QUIT logic

If you have custom code in these areas, renumber it to an unused area.

5. Merge the main conversion lines and any custom code from your existing conversion program into the skeleton. Generally this is lines 2000–3999 and any other custom routines that do not fall into the ranges of code used by the skeleton program.

6. Change file opening logic as follows:

old: GOSUB 5200

new: GOSUB 4500.

7. Change file error message logic as follows:

old: GOSUB 5695

new: LET LOG=1; GOSUB 4800

8. Change any other messages that have been used on the error log, based on the chart above under Method 1.

9. To copy files, GOSUB 4200 and use the following logic for each file:

```
LET F$="<filename>"+STBL("CID"); GOSUB COPYFILE
```

10. Change line 9997. For the last program in the chain (the current version),

```
LET PGM1$="RMCONVER"
```

to let the driver know that the application is finished. For all other programs in the chain,

```
LET PGM1$=APPLDIR$+FILENAME$; REM "the next conversion program in the chain"
```

11. If desired, change any conversion sections to use the new standard for converting a data file. Starting at version 7.5, OSAS introduced a new mechanism for converting a single data file, given 2 templates: the prior version's template, and the current version's template. Use the program "GENCVTTP.PUB to perform this proces. A sample code section follows:

```

0010 LET CHAN=UNT, F$="INVE"+STBL("CID")
1580 REM "----- ARCUX File"
1590 LET F$="ARCU"+COMP$, RETURNLINE=9; GOSUB 4500; IF FAIL THEN GOTO 1720
1600 REM LET F$=DESTDIR$+"ARCU"+COMP$+".EM", TASK=0, F0=75, GENOPEN.CRYP=0;
GOSUB GENOPEN
1610 READ (FILE[OLDTPLCH], KEY="ARCU")*, TPL$; DIM OLDARCU$: TPL$
1620 READ (OSTPLCH, KEY="ARCU")*, TPL$; DIM NEWARCU$: TPL$, DEFAULT$: TPL$,
PAD$: TPL$, ALIGNMENT$: TPL$
1630 RET9:
1640 LET COPY.CHAN1=SOURCECH, COPY.CHAN2=DESTCH
1650 LET DEFAULT.STAT$="A", DEFAULT.RSV5$=FILL(48), DEFAULT.RSV6$=FILL(50)
1660 LET COPY.SPECIAL$=STBL("ARDIR")+ "ARCNVT.750::CUSTOMER_FILE"
1670 CALL "GENCVTTP.PUB", FILE[ALL], COPY$, OLDARCU$, NEWARCU$, DEFAULT$,
PAD$, ALIGNMENT$, MSG$, OSGUI$
1680 IF MSG$<>" " THEN GOSUB ERRORLOG
1690 LET RETURNLINE=10
1700 RET10:
1710 CLOSE (SOURCECH); CLOSE (DESTCH)

```

This code sample works as follows:

- 1590 - Opens up the ARCUX file
- 1610 - Reads the old template from OSTPL.700 and dimensions OLDARCU\$ to that string
- 1620 - Reads the current template from OSTPL and dimensions NEWARCU\$, DEFAULT\$, PAD\$, and ALIGNMENT\$ to that string
- 1640 - Sets the proper channels of the old and new file into the COPY template
- 1650 - Set DEFAULT.STAT\$ (customer status) to "A" (for active), and sets two reserved fields to 48 and 50 spaces respectively
- 1660 - Sets a special program to call when converting each record of the ARCUX file
- 1670 - Calls GENCVTTP.PUB to conver the file

See the section on GENCVTTP.PUB in chapter 3 for a definition on each template used by GENCVTTP.PUB.



Report and Screen Conventions

Report Conventions

Beginning with version 5.00, Open Systems adopted a new standard for reports. The new standards have improved the reports from the standpoint of readability and aesthetics. Follow these standards to produce reports that are consistent in layout. You may run into situations that aren't addressed below; not every situation can be addressed here. Think about what layout will make the report easy to read. You may have a lot of questions when you begin, but you will gradually learn to lay out easy-to-read reports.

A report at the end of these conventions illustrates how the reports should look. The call-out numbers correspond to some of the items listed below. If you have a question about a fine point, the sample report might answer your question.

1. Leave a blank line after the report title and the column headings and between underlined column headings and the first line below the underline.
2. Leave enough space between the word *Page* and the page number for 9999.
3. Use uppercase and lowercase in the report title and column headings consistently. Readability experts say that words in full uppercase are difficult to read.
4. Do not hyphenate the words **on account** when they are used as a noun. Hyphenate them when they are used as an adjective (for example, an on-account balance).
5. Do not separate or hyphenate the word **subtotal**. Do not hyphenate or join the words **grand total**. (If you capitalize the word **grand**, capitalize the word **total** too).

6. When the words **total**, **grand total**, and **subtotal** are printed at the bottom of a report, put them in full caps and line them up with the left margin of the column that is closest to the total, grand total, or subtotal. Use only the singular form of these words.
7. Place the current system date and the run time in the upper left corner of the report. Do not indent the run time below the system date. Use the format 02/02/1995 (not 2/2/1995 or any other variation) for the date. Use the format 1:30 PM (not 01:30 PM) for the run time.
8. Put a double underline (equal signs) above grand totals at the bottom of a column. Do not leave a space above and below the line.
9. Do not present grand totals unless you have presented totals.
10. Space columns evenly so that there aren't large gaps between some columns and data is not crowded together in others.
11. Do not put spaces between letters of a word. Readability experts say that such spacing makes the word difficult to read.
12. Do not use symbols or abbreviations in titles.
13. Do not put a space before and after a slash (/). Use slashes only in dates. Do not use them in titles, column headings, and names of functions; for example, instead of naming a function Makes/Models, name it Makes and Models.
14. Leave one space before and after an equal sign (=).
15. If a column heading spans more than one column, center the heading over the columns and put a line of hyphens before and after it to span the columns that the heading covers. Leave a space between the hyphens and the letters, but do not put spaces between the hyphens.

---- Regular ----
Hours Amount

If only one hyphen fits before and after the column heading to fill out the space, leave it out.

16. Put a single line (hyphens) under column headings. If the columns run straight across the page, run the line straight across the page without breaking it for columns.
17. Make sure that the user knows what kind of value is in a column. Distinguish between **Amount** (monetary amount), **Percentage**, **Unit**, **Quantity**, **Rate**, and so on. Because you cannot line up dollar signs, don't use them in **Amount** columns.
18. The phrase **End of Report** should be printed at the left margin without any markings (asterisks, hyphens, whatever) before and after it.
19. When you abbreviate a word in a report, put a period after the abbreviation. When you abbreviate a word on a screen, do not put a period after the abbreviation.
20. Use colons (:) sparingly. You should have little reason to use a colon in a report. Do not use a colon between a field and its entry. For example, do not insert a colon between the **Phone** field and the phone number.

Use hyphens for single underlines and hyphenated words. Generally, do not use hyphens in report and screen titles and column headings. Use a hyphen in a title only when it doesn't make sense without a hyphen.

The report example below illustrates the use of these rules:

09/19/95 1:30 PM		Builder's Supply Miscellaneous Payroll Journal By Transaction				Page 9999	
Employee ID	Name	Date	Dept.	Description	-- Transaction -- Hours	Amount	
BOU001	Bourne, Linda C	08/20/1995	500	PAY 02 Travel Exp	.000	1500.00	
BOU001	Bourne, Linda C	08/20/1995	500	PAY Vacation	40.000	.00	
GER001	Gerard, Timothy G	08/20/1995	500	PAY 05 Bonus	.000	300.00	
JEN001	Jenkins, K M	09/01/1995	500	DED 09 Cash Adv	.000	700.00	
JEN001	Jenkins, K M	09/01/1995	500	PAY 02 Travel Exp	.000	750.00	
JON001	Joachim, M K	09/13/1995	500	DED 08 Parking	.000	15.00	
JON001	Joachim, M K	09/13/1995	500	DED 07 IRA Plan	.000	65.00	
ROS001	Rossini, L A	09/13/1995	500	PAY Sick	16.000	92.00	
SUBTOTAL				PAY 02 Travel Exp	.000	2250.00	
				PAY 05 Bonus	.000	300.00	
				PAY Vacation	40.000	.00	
				PAY Sick	16.000	92.00	
				DED 07 IRA Plan	.000	65.00	
				DED 08 Parking	.000	15.00	
				DED 09 Cash Adv	.000	700.00	
						=====	
GRAND TOTAL PAY						2642.00	
GRAND TOTAL DED						780.00	
End of Report							

Screen Layout Conventions

Follow these standards and naming conventions when you lay out the report pick screen:

1. Place the menu path name at the top of each screen.
2. Do not assign the same name to more than one function, screen, or report.
3. Use function names on menus that show the relationship between functions. For example, call the function on the **File Maintenance** menu **Sales Reps**, and call the related list **Sales Reps List** on the **Master File Lists** menu.
4. Make the items on menus parallel. For example, on the **Transaction Type** menu, use the names **Create Recurring Entries** and **Edit Recurring Entries**, not **Recurring Entries** and **Edit Recurring Entries**.
5. Do not use abbreviations for menu items.

6. Call items on a Reports menu **reports** or **journals**. Confine lists to the **Master File Lists** menu. Apply the term **log** only to automatically generated “reports.”
7. Make sure that the function name, the title of the screen, and the title of the report match (exactly).
8. Do not use celebrity names in the sample data.
9. Use terms on screens and in reports that are intuitive to the user.
10. Use the word **Append** when something can be added only to the end of something else. Use the word **Add** when something can be added anywhere.
11. When there is enough space, write messages as complete sentences, and punctuate them as you punctuate sentences. If you must use a fragment sentence because space is limited, punctuate it the same way you would punctuate a complete sentence.
12. Use questions marks after fields only if they have a **YES/NO** response.
13. On command bars, highlight the active letter of each command. Capitalize only the active letter of each command. Put a comma between commands.
14. Use the following format for masks:

x-x characters (for alphanumeric fields)

1-3 digits (for numeric fields)

x.x (for fields that use decimal points)
15. Do not use the word **Pick** on screens that have only one pick option. When you use the word **Pick**, line up the options under the first one so that **Pick** stands alone to the left of the first option.
16. Use the expression “Do You Want to Maintain” instead of “Do You Want to Add/Change” when the user has a choice of screens.

Accepted Abbreviations

account	acct	General Ledger	GL
accounts payable	AP	gross	gr
accounts receivable	AR	hours	hr
acquisition	acq	identification	ID
additional	addnl	include	include
adjustment	adj	insurance	ins
alphanumeric	alphnum	Inventory	IN
amount	amt	invoice	inv
approximate	approx	Job Cost	JO
available	avbl	manufacturing	mfg
average	avg	maximum	max
balance	bal	merchandise	mdse
Bank Reconciliation	BR	minimum	min
business	bus	month-end	mo-end
cash on delivery	COD	number	no
cent	ct	on hand	oh
characters	char	paid	pd
check	ck	payment	payt
collateral	collat	Payroll	PA
collect	col	percent, percentage	pct
commission	comm	per hour	per hr
company	co	period	per
compensation	comp	Purchase Order	PO
continued	cont	quantity	qty
contribution	contrib	regular	reg
cost of goods sold	COGS	report	rept
cost of living	COL	Report Writer	RW
credit	cr	Resource Manager	RM
current	cur	retirement	ret
customer	cust	Sales Order	SO
debit	db	Social Security	SS
department	dept	standard	std
depreciation	depr	Time Billing	TI
description	desc	transactions	trans
discount	disc	unit of measure	UOM
dollar	dol	warehouse	whse
double	dbl	withholding	wh
each	ea	worksheet	wksht
expense	exp	year-end	yr-end

CHAPTER 3

3

General Routines	3-3
Application-Specific Routines	3-129

Routines

This chapter details the general and application-specific routines. This tables on the first section list the routines and a brief description of each.

General Routines

Routine	Description
GENACDOC.PUB	entry for user-defined documents for a given master file
GENADDL.PUB	entry for Additional Descriptions
GENADDR.PUB	format Name and Address for Postal Regulations
GENBATCH.PUB	entry of batches in AR, SO, AP, and PO
GENCHECK	GUI checkbox selection
GENCHECK_RADIO	GUI radio button selection
GENCMNT.PUB	comment entry for customers, vendors, and jobs
GENCODE.PUB	validate a menu selection for an access code
GENCOPY.PUB	copy a file
GENCOPY2.PUB	copy a single file or a list of files one record at a time
GENCVNUM.PUB	Retrieve/Change check or voucher number from Banks
GENCVTTP.PUB	Converts a data file from template information
GENDATE	noninteractive program interface to GENDATE.PUB
GENDATE.PUB	convert a string date to a Julian number and vice versa
GENDFLT.PUB	retrieve workstation defaults for a terminal
GENEMAIL.PUB	OSAS E-Mail Processor
GENERROR	interactive program interface to GENERROR.PUB
GENERROR.PUB	display program errors, messages and verifications
GENFILE.PUB	create an application data file
GENFPRNT.PUB	print multiple form copies according to the options in the forms code file
GENGETOP.PUB	return the option string for the application specified
GENGETSS.PUB	retrieve next available sort sequence number
GENGOTO.PUB	handle prompting for GOTO commands in HLE screens

Routine	Description
GENGUFNC.PUB	build GUI menu strings and standard GUI tool bars
GENGUHLP.PUB	display help for a GUI data field
GENGUINP.PUB	control input for a data field while in GUI mode
GENGULIB.PUB	library of routines used while in GUI mode
GENGULIN.PUB	control scroll area (and accept single-character commands) in GUI mode
GENGUOTH.PUB	run programs from the Other Commands menu in GUI mode
GENGURES.PUB	display graphical screens from BRC files
GENGUSER.PUB	retrieves a user name from OSUS given a terminal ID
GENGUUSR.PUB	Runs program from the GUI User Menu
GENGUWIN.PUB	display an inquiry window in GUI mode
GENHELP.PUB	display help for a data field in text mode
GENINFO (INFO)	interactive program interface to GENINFO.PUB
GENINFO.PUB	display xxINFO menu and CALL the program task selected
GENINPUT	interactive program interface to GENINPUT.PUB and GENGUINP.PUB
GENINPUT.PUB	control input for a data field in text mode
GENLABEL.PUB	Set up and print labels
GENLINE	interactive program interface to GENLINE.PUB and GENGULIN.PUB
GENLINE.PUB	control scroll area (and accept single-character commands) in text mode
GENLIST	select a list button option in GUI mode
GENLOCK	noninteractive program interface to GENERERROR.PUB
GENLOG.PUB	User Logging functions
GENMAINT.PUB	interface program for the File Maintenance key
GENMPATH.PUB	Determine and change to modifications directory
GENNUM	mask display and error 43 trapping
GENOPEN	program interface to GENOPEN.PUB
GENOPEN.PUB	OPEN or CLOSE a data file

Routine	Description
GENOTBBJ.PUB	BBj Printer Output
GENOTHER.PUB	perform the Other Commands function in text mode
GENOUT.PUB	prompt for output destination
GENPC	program interface to GENPC.PUB
GENPC.PUB	perform period conversion
GENPHONE	phone mask display
GENPRBBJ.PUB	General printer selection for BBj/Java Printers
GENPRNT.PUB	prompt for a printer selection
GENPSCR.PUB	perform a print screen
GENREPL.PUB	replace user-defined text in prefix
GENSAVE.PUB	save workstation environment settings
GENSCROL.PUB	manage on-screen report output
GENSELIN.PUB	special input program for print routines
GENSORT.PUB	manage sort files
GENSORT2.PUB	get a sort order from a list
GENSTDOC.PUB	Start up a document file
GENTAX.PUB	calculate sales tax
GENTAX2.PUB	post calculated sales tax
GENUSR.PUB	Runs programs from the F4 User Menu
GENUSRFM.PUB	entry for user-defined fields for a given master file
GENVWDOC.PUB	view the user-defined documents attached to a given master file
GENWIND.PUB	display an inquiry window in text mode
GENWKS.PUB	format output for a Lotus file

Routines ending with the extension .PUB are public programs. Access them with a CALL statement to the routine name in quotation marks:

```
xxxxx CALL "GENCODE.PUB", . . .
```

Routines with no extension are general routines. You must merge them into a program and then access them with a GOSUB statement to the routine name (a label):

```
xxxx GOSUB GENINPUT
```

The general library (GENLIB) file included with the Software Development Utilities contains the general routines. You can use this file with the BASIS library manager to merge or update one or more general routines in one or more programs.

GENACDOC.PUB

GENACDOC.PUB is a public program that allows the user to enter in user defined documents for a given master file. Use this program if you have set up a separate file that holds documents for any master file you have.

```
0055 ENTER FILE[ALL],USER_ACDOC$,XXUD$,OSGUI$,PROMPT_NUM
```

Variable	Description
FILE[ALL]	Standard file array.
USER_ACDOC\$	Template containing setup information for this function.
XXUD\$	The template for the base application's xxUD file.
OSGUI\$	GUI Template (see GENGURES.PUB on page 3-57 for more detail).
PROMPT_NUM	Number of screen prompts for key fields

Exit Points	Explanation
EXIT	GENACDOC.PUB does not return any errors.

This chart breaks down the USER_ACDOC\$ template:

Field	Description
DOC:N(2)	FILE[x] array element on which xxUD file is opened.
TITLE:C(70)	Title the add/change screen displays.
LABEL[x]:C(20)	Label that appears for each prompt.
ID[x]:C(20)	Data that appears for each prompt.
DESCR[x]:C(35)	Description that appears for each prompt.
TASK:C(1)	Passes back Y\$ to the calling program.
TYPE:C(1)	“N” to have screen with Next/Previous Screen prompts, or “D” to have screen with Done prompt.

Note: The number passed in PROMPT_NUM determines how many prompt fields appear on the screen, and is used to prepare the USER_ACDOC\$ template.

When setting up to use this program, you can call GENACDOC.PUB to have the USER_ACDOC\$ template set up for you. The template USER_ACDOC\$ has its arrays dimensioned to the number you pass in PROMPT_NUM.

```
5905 SET_TEMPLATE:  
5907 SETERR GENERROR  
5910 ENTER USER_ACDOC$,PROMPT_NUM
```

Variable	Description
USER_ACDOC\$	Template containing setup information for this function.
PROMPT_NUM	Number of screen prompts for key fields

GENADDL.PUB

GENADDL.PUB is a public program that allows the user to enter up to 10 lines of additional descriptions consisting of 35 characters each. You may use this program anywhere you need to prompt for additional descriptions.

```
0090 ENTER XT$[ALL],STARTX,STARTY,TASK$,START,OSGUI$
```

Variable	Description
XT\$[1:10]	Array that holds additional descriptions. Array must be pre-dimensioned and populated with descriptions before calling program.
STARTX	X1 coordinate used by GENINPUT.PUB to enter line items
STARTY	X0 coordinate used by GENINPUT.PUB to enter line items
TASK\$	Returns “U” if user used the Up command from the first line of descriptions.
START	Line on which to begin entering additional descriptions
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENADDL.PUB does not return any errors.

GENADDR.PUB

GENADDR.PUB is a public program that properly formats a given name and address to a given number of lines based on US Postal Service regulations. Use this program where you print address lines that require adherence to USPS rules.

```
0100 ENTER FILE[ALL] , NA$, ADDRESS$[ALL] , LINES, WIDTH, MSG$
```

Variable	Description
FILE[ALL]	Standard file array
NA\$	Template containing information to be formatted
ADDRESS\$[ALL]	Array containing properly formatted name and address
LINES	Number of lines to be returned to the calling program
WIDTH	Number of characters wide the address lines need to be
MSG\$	Error message returned to the calling program

Exit Points	Explanation
EXIT	GENADDR.PUB does not return any errors.

This chart breaks down the NA\$ template:

Field	Description
ATTN:C(50)	Attention
NAME:C(50)	Name
ADDR1:C(50)	Address Line 1
ADDR2:C(50)	Address Line 2
ADDR3:C(50)	Address Line 3
CITY:C(50)	City
STATE:C(3)	State
ZIP:C(12)	Zip Code
CTRY:C(2)	Country
PRTNAME:N(1)	Boolean whether or not to print name in address lines

GENBATCH.PUB

GENBATCH.PUB is a public program that prompts for batches where necessary in Accounts Payable, Accounts Receivable, Purchase Order, and Sales Order. There are four separate entry point that are available for use depending on how you need to work with batches.

BATCHES:

This entry point prompts for batches to be tagged in preparation for printing reports and posting batches.

```
0055 BATCHES:
0060 ENTER FILE [ALL] , BATCH_TMPL$ , BATCH$ , COUNT , Y$ , PARAM , OSGUI$ , O2$
```

Variable	Description
FILE[ALL]	Standard file array
BATCH_TMPL\$	Template containing information for batch entry
BATCH\$	String of batches that are selected by the user
COUNT	Total number of batches selected
Y\$	Standard command string
PARAM	Variable used in restart logic for restart testing
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57)
O2\$	Application options of calling application

Exit Points	Explanation
EXIT	GENBATCH.PUB does not return any errors.

This chart breaks down the BATCH_TMPL\$ template:

Field	Description
JRNL1:C(7)	Report printing status, column 1 title; may be blank
JRNL2:C(7)	Report printing status, column 2 title; may be blank
JRNL3:C(7)	Report printing status, column 3 title; may be blank

Field	Description
APPL:C(2)	Calling application ID
FILE:N(2)	FILE[x] array element on which xxBTx file is opened
FLAG_RPT:C(1)	Flag to set reports to when printed
POST:N(1)	Set when using transaction posts

SET_TEMPLATE:

This entry point sets up the template BATCH_TMPL\$.

```
10005 SET_TEMPLATE:
10010 ENTER BATCH_TMPL$
```

NO_BATCHES:

This entry point is used when the calling application is not using batching as an option for transaction entry. This updates the blank batch record's report statuses and lock flags as necessary.

```
12002 SET_TEMPLATE:
12005 ENTER FILE [ALL] , BATCH_TMPL$ , BATCH$ , COUNT , Y$ , PARAM , OK , O2$ , OSGUI$
```

UNFLAG_BATCHES:

This entry point resets the lock flag on the batches that were initially locked by GENBATCH.PUB. Those batches are contained in the BATCH\$ string.

```
13005 UNFLAG_BATCHES:
13010 ENTER FILE [ALL] , BATCH_TMPL$ , BATCH$
```

GENCHECK

GENCHECK is a general routine that contains a multiline function called FNCHECK(). This function takes a control ID and a string, and toggles a GUI check box control between checked and unchecked states based on the first character in the string. GENCHECK occupies lines 18800–18849 and should be merged into all programs that need to manipulate check boxes while in GUI mode.

Variable	Description
ID	GUI control ID of the check box
PARAM\$	String to check for 'Y' as the first character

To use GENCHECK, add a line in the GUI_DATA section where a YES/NO answer would normally appear in the DATA section of code. For example, if the code in the data block prints YES/NO data like this:

```
7720 PRINT @(22,10),PRINT_DETAIL$,
```

Its GUI equivalent would read

```
17720 LET TMP=FNCHECK(4100,PRINT_DETAIL$)
```

There may also be situations where the YES/NO in text is based on a function that looks at a boolean value, such as:

```
7720 PRINT @(22,10),FNYESNO$(PRINT_DETAIL)
```

The GUI equivalent would read

```
17720 LET TMP=FNCHECK(4100,FNYESNO$(PRINT_DETAIL))
```

The FNYESNO\$() function will evaluate to a YES/NO string, which can then be used in FNCHECK().

GENCHECK_RADIO

GENCHECK_RADIO is a general routine that contains a multiline function called FNCHECKRADIO(). This function takes a numeric value and a string, and will select a radio button from a provided list, based on the numeric value. GENCHECK_RADIO occupies lines 18850–18890 and should be merged into all programs that need to manipulate radio buttons while in GUI mode.

Variable	Description
PARAM	Numeric value on which to base selection
LIST\$	List of control ID's of radio buttons, each 5 characters long, with no separator between them

Use radio buttons only on screens where a 1-x selection is offered on character screens. Examples of these would be on report pick screens and transaction entry type selection.

To use GENCHECK_RADIO, add a line in the GUI_DATA section where a numeric value would print in the DATA section of code. For example, if the code in the data block like this:

```
7720 PRINT @(22,10),STR(TRANSACTION_TYPE:"#"),
```

Its GUI equivalent would be as follows:

```
17720 LET TMP=FNCHECK_RADIO(PRINT_DETAIL,"041010410204103")
```

Notice that all radio controls are passed to the function. The function determines which radio button to check based on the value of PRINT_DETAIL.

GENCMNT.PUB

GENCMNT.PUB is a public program that is used for both comment entry and comment inquiry. Currently customers, vendors, and jobs support comments. The comment file structures for these three are identical with the exception of the jobs comment file, because both jobs and their phases may have comments. The program contains two entry points.

COMMENT:

This entry point will prompt for comment entry and provide inquiry function for comments as well.

```
0050 COMMENT:
0060 ENTER FILE[ALL], MASTER$, COMMENT$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
MASTER\$	Template containing the master file record for the selected customer, vendor, or job
COMMENT\$	Template containing the information for comment entry
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENCMNT.PUB does not return any errors.

This chart breaks down the COMMENT\$ template:

Field	Description
CMNT:N(2*)	FILE[X] index on which the comment file is open
MASTER:N(2*)	FILE[X] index on which the master file (customer, vendor, or job file) is open
TYPE:C(6*)	Used for displaying the type of comment (“Customer”, “Vendor”, or “Job”)

Field	Description
FULL:C(1*)	Set to Y for full-screen display, or to N for a short screen. Full screens are used in maintenance functions, short screens are used in inquiry functions.
MAINT:C(1*)	Set to Y to allow for editing of comments, or to N to prohibit editing
INQUIRY:C(1*)	Set to Y to allow inquiry on comments, or N to prohibit inquiry
STARTROW_N(1*)	Starting row where comment box will be displayed (used only if COMMENT.FULL\$ is set to N)
ID:C(6*)	The master file's ID calling program is working with. This will be the customer, vendor, or Job ID.
IDNAME:C(6*)	The template field name within MASTER\$ where the ID can be found
NAME:C(6*)	The name associated with the comment ID (for example, customer name, vendor name, or job name)
WINDID:C(9*)	Window ID used for main ID (customer, vendor, or job)
SELVAL:C(10*)	Used in conjunction with WINDID\$

SET_TEMPLATE:

This entry point will set the template COMMENT\$.

```
5905 SET_TEMPLATE:
5910 ENTER COMMENT$
```

GENCODE.PUB

GENCODE.PUB is a public program that verifies access to a particular program or menu for the current access code. Use this program in file maintenance programs called by GENMAINT.PUB.

```
0090 ENTER TYPE$,NAME$,PARAM$,ACCESS
```

Variable	Description
TYPE\$	Defines the type of entity in NAME\$: "1" - menu file "2" - menu record "3" - RUN program "4" - CALLED program "5" - SCALLED program "6" - CALLED information window program
NAME\$	Program, menu file or menu record as defined by TYPE\$
PARAM\$	Parameter passed to a program, or the menu file if TYPE\$ is 2, a menu record
ACCESS	Set to 0 if access is granted. Set to 1 if access is denied.

Exit Points	Explanation
EXIT	GENCODE.PUB does not return any errors.

GENCOPY.PUB

GENCOPY.PUB is a public program for copying a source file to a destination file. Each parameter must include the path location and name of the file.

```
0100 ENTER SOURCE$,DEST$
```

Variable	Description
SOURCE\$	Location and name of source file
DEST\$	Location and name of destination file

Exit Points	Explanation
EXIT	GENCOPY.PUB does not return any errors.

GENCOPY2.PUB

GENCOPY.PUB is a public program for copying a source file to a destination file. The file is copied reading and writing one record at a time, so that an existing Mkeyed file can be converted to a highly-recoverable Mkeyed file. Each parameter must include the path location and name of the file.

```
0100 ENTER SOURCE$,DEST$,CID$,APPL$,XXFILE$,OSGUI$
```

Variable	Description
SOURCE\$	Location and name of source file(s)
DEST\$	Location and name of destination file(s)
CID\$	Company ID of current company
APPL\$	Current application ID
XXFILE\$	Name of xxDATA.610 file to use to create the new file(s)
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENCOPY.PUB does not return any errors.

If SOURCE\$ starts with the word LIST, then a list of files may be included within SOURCE\$, each separated by a comma. The destination path is then set in DEST\$.

GENCVNUM.PUB

GENCVNUM.PUB is a public program for retrieving and changing next check or voucher numbers from banks. Since we store the next check and voucher numbers with specific banks, a general interface was developed so that the next available check and voucher numbers are changed in a consistent manner.

```
0050 ENTER FILE[ALL] , TYPE$ , PROCESS$ , BANK$ , NUMBER , MSG$
```

Variable	Description
FILE[ALL]	Standard file array
TYPE\$	"C" = Get/Change Check number "V" = Get/Change Voucher number
PROCESS\$	"G" = Retrieve check or voucher number "W" = Write check or voucher number
BANK\$	Bank ID to process
NUMBER	Check or voucher number
MSG\$	Error message if bank is not found

Exit Points	Explanation
EXIT	GENCOPY.PUB does not return any errors.

GENCVTTP.PUB

GENCVTTP.PUB is a public program that performs several functions. It's key function is to convert data files using previous and current template information.

(No Entry Label)

This entry point is the main entry point for accessing this public routine.

```
0060 ENTER FILE[ALL], COPY$, TEMPLATEOLD$, TEMPLATENEW$, DEFAULT$, PAD$,  
ALIGNMENT$, MSG$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
COPY\$	Template with conversion information
TEMPLATEOLD\$	Dimensioned to old file template being converted
TEMPLATENEW\$	Dimensioned to new file template being converted
DEFAULT\$	Dimensioned to new file template being converted. Individual fields can be set to defaults values and destination file will contain those defaults
PAD\$	Dimensioned to new file template being converted. Individual fields can be set with a pad character to pad the destination field with the character. Leaving this blank sets default pad character to " "
ALIGNMENT\$	Dimensioned to new file template being converted. Individual fields can be set to force a padding method with the PAD character from the PAD template. "Z" = Zero padded (ignores PAD character) "L" = Left padded using PAD character "R" = Right padded using PAD character
MSG\$	Returns error messages
OSGUI\$	Standard GUI template

Exit Points	Explanation
EXIT	GENCVTTP.PUB does not normally return an error.
EXIT ERR	GENCVTTP.PUB returns untrapped errors if one occurs.

Normal errors will be handled thru ERRORTRAP function, which like most application conversion programs, will temporarily halt the conversion process if the user desires so they can break to console mode and attempt to debug the source of the error.

This chart breaks down the COPY\$ template:

Field	Description
FILE1NAME:C(12*)	Source file records are read from
FILE2NAME:C(12*)	Destination file records are written to
CHAN1:N(2*)	Channel number source file is opened on. If set to 0, COPY.FILE1NAME\$ will be opened via TASK=7 thru GENOPEN.PUB.
CHAN2:N(2*)	Channel number source file is opened on. If set to 0, COPY.FILE2NAME\$ will be opened via TASK=7 thru GENOPEN.PUB.
SPECIAL:C(50*)	Program name called on every record read
LOGCH:N(2*)	Channel that error log is opened on
WARN:C(1*)	“Y” or “N” based on users response on warning if any errors occur during conversion
O9:C(504*)	Set to O9\$ is doing any date conversion with GENDATE.PUB
PPY:N(2*)	Set to number of periods per year
EXTRASTR[5]:C(50*)	Array of extra strings carried thru to called program found in CALL.SPECIAL\$
EXTRANUM[5]:N(2*)	Array of extra numerics carried thru to called program found in CALL.SPECIAL\$

INITIALIZE_TEMPLATE:

This entry point sets the CALL\$ template. This template is used to provide GENCVTTP.PUB with general information regarding the conversion process.

```
4305 INITIALIZE_TEMPLATE:  
4310 ENTER COPY$
```

Variable	Description
COPY\$	Standard conversion template

GENDATE

GENDATE is a general routine that calls GENDATE.PUB to validate a date. GENDATE occupies lines 9000–9099. Merge it into a program only if the program must validate a date field without using GENINPUT.PUB. Most programs should use GENINPUT.PUB instead of calling GENDATE.PUB directly.

Variable	Description
X\$	Date in string format. The date string is reformatted after it is truncated.
X1\$	Date mask for date
X0	Row of date field on screen (ignored if X3 is not 2)
X1	Column of date field on screen (ignored if X3 is not 2)
X3	Input field type, normally 2 for string with a mask
X	Julian date of X\$. These values are returned if a problem is found: -1 - a blank date -9 - an invalid date

GENDATE.PUB

GENDATE.PUB is a public program for converting dates in string format to a Julian number. GENDATE.PUB is called by GENINPUT.PUB when date fields are used as input (GENINPUT with X3=6).

```
0040 ENTER X$,X1$,X,X0,X1,X3
```

Variable	Description
X\$	Date as defined in GENDATE
X1\$	Date mask for date
X	Julian date of X\$. These values are returned if a problem is found: -1 - a blank date -9 - an invalid date
X0	Row of date field as defined in GENDATE
X1	Column of date field as defined in GENDATE
X3	Input field type as defined in GENDATE

Exit Points	Explanation
EXIT	GENDATE.PUB does not return any errors.

GENDATE.PUB translates the date string into a Julian number. The global variable "DATEMASK" is used to interpret the format of the date string. The date string is then reformatted and is redisplayed on the screen if X3 is set to 2 for string input with a mask.

GENDFLT.PUB

GENDFLT.PUB is a public program for retrieving workstation default information for the current terminal.

```
0100 ENTER OSDF,TERM$,USER$,DF1$,DF2$,DF3$[ALL],DF4$[ALL],STAT
```

Variable	Description
OSDF	The FILE[] array index on which OSDF is open
TERM\$	The terminal ID of the terminal
USER\$	The user name for the terminal
DF1\$,DF2\$, DF3\$[ALL], DF4\$[ALL]	OSDF record data
STAT	The BBx error that occurred, if any

Exit Points	Explanation
EXIT	GENDFLT.PUB does not return any errors.
EXIT ERR	Sets STAT to the BBx error and EXITs.

GENDFLT.PUB reads the OSDF file using the terminal ID supplied, sets USER\$, and returns in the DFx\$ variables the data read.

GENEMAIL.PUB

GENEMAIL.PUB is a public program that performs several functions related to e-mail processing.

GET_INFORMATION:

This entry point prompts the user for from, to, CC, subject, and attachment information when sending e-mails when users select the EMAIL option when processing reports.

```
1000 GET_INFORMATION:
1010 ENTER MSG$,OSGUI$
```

Variable	Description
MSG\$	Set if unable to open file OSED
OSGUI\$	Standard graphical template

Exit Points	Explanation
EXIT	This callpoint does not normally return an error.

SEND_EMAIL:

This entry point will take STBL variables that were set when the GET_INFORMATION call point was called, and send an e-mail from OSAS

```
2000 SEND_EMAIL:
2010 ENTER MSG$,OSGUI$
```

Variable	Description
MSG\$	Contains any error messages when trying to process e-mail from mailcall.bb
OSGUI\$	Standard graphical template

Exit Points	Explanation
EXIT	This callpoint does not normally return an error.

PROCESS_EMAIL_QUEUE:

This entry point will take all e-mails in the OSQA and OSQE files, for the user ID found in STBL("USERID") and attempt to send them.

```
3000 PROCESS_EMAIL_QUEUE:
3010 ENTER MSG$,OSGUI$
```

Variable	Description
MSG\$	Contains any error messages when trying to process e-mail from mailcall.bb
OSGUI\$	Standard graphical template

Exit Points	Explanation
EXIT	This callpiont does not normally return an error.

PROCESS_EMAIL_FROM_MENU:

This entry point, which is called from all menu types, will check if any e-mails exact for the current user ID found in STBL("USERID"), and attempt to send them.

```
4500 PROCESS_EMAIL_FROM_MENU
45010 ENTER MSG$,OSGUI$
```

Variable	Description
MSG\$	Contains any error messages when trying to process e-mail from mailcall.bb
OSGUI\$	Standard graphical template

Exit Points	Explanation
EXIT	This callpiont does not normally return an error.

GENERERROR

GENERERROR is a general routine that interfaces to GENERERROR.PUB for handling program errors, messages and verifications. The GENERERROR2 routine is also required. GENERERROR and GENERERROR2 occupy lines 8400–8499 and should be merged into all interactive programs.

Variable	Description
X9\$	Message to display before waiting for input. X9\$ is limited to 76 characters and is cleared before GENERERROR.PUB exits.
X8\$	Message to display without waiting for input unless coupled with an X9\$ message or O7 value, in which case X8\$ serves as a second message line. X8\$ cannot exceed 76 characters. Clear it by passing X8\$=" " to GENERERROR. If X8\$ is coupled with a X9\$ or O7 value, it is cleared automatically by GENERERROR.
O7	Selects a predefined message or verification prompt. -3 - an error occurred (used by GENERERROR only) -2 - an ESCAPE occurred (used by GENLOCK only) -1 - an error occurred (used by GENLOCK only) 0 - no error and no message selected 1 - Go to Menu 2 - Proceed 3 - Delete 4 - Abandon 5 - Invalid Date 6 - Invalid Entry 7 - Record Not Found 8 - 'xxxxxxx' is not on file 10 - Prompts for a Yes/No choice in GUI mode only. May populate other choices besides Yes/No. To do so, place choices in X\$ separated by a \$0D\$ O7 is set to 0 before exiting. Use X8\$ to add a second message line. Use X9\$ to add a second message line and force the user to press the key for the O7 value (1–4 are the only key-related values and only 1 makes sense). X8\$ and X9\$ cannot be used with an O7 value.

Variable	Description
Y\$	Command to be processed from GENINPUT. A non-blank X8\$ or X9\$ value or a non-zero O7 value take precedence over a Y\$ value. Y\$ is set to the appropriate value after a key is pressed before GENERERROR.PUB exits.
MX8	Numeric index for an X8\$ message stored in an xxMS (application message) file. Used in lieu of X8\$.
MX9	Numeric index for an X9\$ message stored in an xxMS (application message) file. Used in lieu of X9\$.
X1\$	Mask that caused a mask format error
F\$	File name that caused a file error
PROGID\$	Program to call for the File Maintenance key
FILE[ALL]	Standard file array
YDISABLE\$	String of Y\$ values that should be ignored. Note that only ESC sequence commands can be disabled (that is, Esc+X for Abandon, but not Ctrl+K for Up).
PGM(-2)	Name of program calling GENERERROR.PUB
X\$	Used in certain O7 values. Mainly will be used in O7=8 or O7=10 for GUI
X4	Length of input from GENINPUT.PUB
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
RETURN	GENERERROR normally returns.
RETRY	A RETRY is attempted if Y\$ is "R".
GOTO 8470	Y\$ = "S" for a Start Over key press.
GOTO 8480	Y\$ = "X" for an Abandon key press (and verified if necessary).
GOTO 9990	Y\$ = "M" for an Exit key press (and verified if necessary).

Part of GENERERROR is designed to handle BASIC errors. To take advantage of this feature, place a SETERR GENERERROR statement at line 1 of a program. GENERERROR can trap and interpret errors. It displays appropriate error messages and determines whether the user should be allowed to return to the program, retry the operation that failed, or be forced to exit to the menu.

The remainder of GENERERROR is designed to display messages based on the values of X8\$ or MX8, X9\$ or MX9, O7 and/or Y\$, and wait for the user to acknowledge the message (with the exception of X8\$).

In both cases GENERERROR exits based on the value returned in Y\$ from GENERERROR.PUB.

GENERERROR.PUB

GENERERROR.PUB is the public program for handling errors, messages and verifications. The interface programs GENERERROR or GENLOCK should always be used to access GENERERROR.PUB.

```
0100 ENTER ERR_MSG$,MSG$,CUR_ERR,ERR_LINE,VERIFY,Y$,HOST_ERR,MASK$,FILE$,
0100:PROGID$,FL[ALL],YDISABLE$,PGM$,XKEY$,MX4,M8,M9,OSGUI$
```

Variable	Parameter	Description
ERR_MSG\$	X9\$	Message as defined in GENERERROR
MSG\$	X8\$	Message as defined in GENERERROR
CUR_ERR	ERR	Last BASIC error
ERR_LINE	TCB(5)	The line on which the error occurred
VERIFY	O7	Predefined message as defined in GENERERROR
Y\$	Y\$	Command as defined in GENERERROR
HOST_ERR	TCB(10)	Last host system error
MASK\$	X1\$(1)	Mask variable assigned for use by GENINPUT.PUB
FILE\$	F\$(1)	The data file name to be OPENed by the GENOPEN routine. It will be displayed when a file operation cannot be performed.
PROGID\$	PROGID\$	Program to call for the File Maintenance key
FL[ALL]	FILE[ALL]	Standard file array
YDISABLE\$	YDISABLE\$	See definition of YDISABLE\$ for GENERERROR
PGM\$	PGM(-2)	Current program name
XKEY\$	X	Current field value
MX4	X4	Field length
M8	MX8	Numeric index to xxMS application message file
M9	MX9	Numeric index to xxMS application message file

Variable	Parameter	Description
OSGUI\$	OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENERROR.PUB does not exit with an error code.

GENERROR.PUB determines the appropriate message(s) to display, waits and validates an acknowledgment if necessary, and exits with the appropriate command in Y\$.

GENFILE.PUB

GENFILE.PUB is a public program used to create a single file based on the information stored in `xxDATA.yyy`. This routine is used to make sure that a file's key definition will be created using the correct key information for a particular application.

```
0100 ENTER COMP$,APPLCODE$,DESTDIR$,DATANAME$,ERRCODE,ERRMSG$,OSGUI$
```

Variable	Description
COMP\$	Company ID
APPLCODE\$	Application ID
DESTDIR\$	Directory in which to create the file
DATANAME\$	Data record key for the file in <code>xxDATA.yyy</code>
ERRCODE	Error number returned if file create fails
ERRMSG\$	Error message if file create fails
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENFILE.PUB normally exits with no error. If an error did occur, its value is returned in the variable <code>ERRCODE</code> .

GENFPRNT.PUB

GENFPRNT.PUB is a public program that prints multiple copies of a form using the information stored in the forms codes file.

```
0060 ENTER COPIES,FORM_TYPE$,PRINT_DEVICE$,PRINT_CHANNEL,DEVICE$,CHANNEL,OSGUI$
```

Variable	Description
COPIES	The number of copies of the form to print
FORM_TYPE\$	The Form Code ID for the form being printed
PRINT_DEVICE\$	The device ID for the printer
PRINT_CHANNEL	The channel number on which the print device is OPENed
DEVICE\$	The name of the temporary print spool file
CHANNEL	The channel number on which the temporary file is OPENed
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENFPRNT.PUB does not return any errors.

GENGETOP.PUB

GENGETOP.PUB is a public program that returns the options string for the application you specify.

```
0030 ENTER APP$,MSG$,OPT$
```

Variable	Description
APP\$	The 2-character application ID whose options you want returned
MSG\$	A message returned by GENGETOP.PUB if the options table or tables file is missing for the requested application
OPT\$	The application's option string

Exit Points	Explanation
EXIT	GENGETOP.PUB does not normally return an error.
EXIT ERR	GENGETOP.PUB returns an error if one occurs.

GENGETSS.PUB

GENGETSS.PUB is a public program that retrieves and returns to the CALLing program the next available sort sequence number that is used to create scratch files in version 7.0 and higher.

This routine is CALLED from both RMSETUP and RMRUNAP to ensure that the scratch files created in programs run from the MDI menu are uniquely named.

```
0100 ENTER SEQ$,ERRMSG$
```

Variable	Description
SEQ\$	The 3-character sequence number to be used in naming scratch files.
ERRMSG\$	A message returned by GENGETSS.PUB if an error occurs during processing.

Exit Points	Explanation
EXIT	GENGETSS.PUB does not return an error value.

GENGOTO.PUB

GENGOTO.PUB is a public program that pops up a box and prompts the user for a string or numeric value. Generally this is used when prompting for go-to boxes within HLEs, when more than one screen of information is displayed. However, because there is no logic to control the HLE, this can be used generically for any type of special prompting where necessary.

For a more detailed definition of the variables used in GENGOTO.PUB, see the section on GENINPUT, GENINPUT.PUB, or GENGUINP.PUB.

```
0060 ENTER Y$, STARTX, STARTY, X2, X3, X4, X9, X1$, X, X$, WINDID$, SELVAL$, COMPID$,
0060:FILE [ALL], YDISABLE$, PGM$, FLD_HLP$, OSGUI$, BUTTON_KEYSS$ [ALL], PROMPT$,
0060:TEXT_COLOR
```

Variable	Description
Y\$...PGM\$	See GENINPUT.PUB variable definitions
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)
BUTTON_KEYSS\$[ALL]	Array for button ID's and the characters to which they tie
PROMPT	Prompt to display in front of the entry box
TEXT_COLOR	Character mode color level at which to display the box

Exit Points	Explanation
EXIT	GENGOTO.PUB does not normally return an error.

GENGUFNC.PUB

GENGUFNC.PUB is a public program that performs several functions that set up GUI screens for OSAS. This program returns various menu strings for use on menu bars, builds the standard tool bar, and sets up the status bar. The program contains six entry points, which all exit from the same exit point.

BUILD_MENU_STRINGS

This entry point returns strings that contain the **Help** menu and the standard menu containing the **Commands**, **Edit**, **Modes**, and **Other** menus. The **Help** menu is commonly the last menu on the menu bar, and is returned in a separate string so that you can insert other menus before the **Help** menu.

```
1020 BUILD_MENU_STRINGS:
1030 ENTER MENUHLP$,MENU1$
```

Variable	Description
MENUHLP\$	String containing the Help menu structure
MENU1\$	String containing the Commands, Edit, Modes, and Other menu structure

Exit Points	Explanation
EXIT	GENGUFNC.PUB does not normally return an error.
EXIT ERR	GENGUFNC.PUB returns an error if one occurs.

BUILD_HLE_MENU:

This entry point returns a string that contains the scroll region menu. It is used by functions that permit movement, **Insert**, **Jump**, and **Delete** within a scroll region.

```
1505 BUILD_HLE_MENU:
1510 ENTER MENU$
```

Variable	Description
MENU	String containing the Scroll Commands menu

DISPLAY_TOOLBAR:

This entry point builds the standard tool bar on the current GUI screen. This entry point is always used by GENGURES.PUB. You may call it separately if necessary.

```
2010 DISPLAY_TOOLBAR:
2020 ENTER OSGUI$
```

Variable	Description
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

BUILD_GRAPHICAL_TOOLBAR:

This entry point builds the standard menu tool bar for the Graphical-style menu only. This entry point is used by RMMENU2 only.

```
2705 DISPLAY_TOOLBAR:
2707 ENTER OSGUI$
```

Variable	Description
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

BUILD_STATUS_BAR:

This entry point builds the standard status bar on the current GUI screen. This entry point is always used by GENGURES.PUB. You may call it separately if necessary.

```
3005 DISPLAY_STATUS_BAR:
3010 ENTER OSGUI$
```

Variable	Description
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

BUILD_ITEM_LOOKUP_MENU_STRINGS:

This entry point returns a string that contains the Item Lookup menu. This menu is used by functions that permit the use of the Inventory lookup functions (Alias Lookup, Customer/Vendor lookup, etc.).

```
3510 BUILD_ITEM_LOOKUP_MENU_STRINGS:
3520 ENTER MENU1$
```

Variable	Description
MENU1\$	String containing the Inventory Lookup menu.

GENGUHLP.PUB

GENGUHLP.PUB is a public program that displays context-sensitive field help in GUI mode. This program is CALLED by GENGUINP.PUB and is invoked when the **Help** key is pressed. All parameters originate from GENINPUT.

```
0100 ENTER HID$,ROW,COL,PAGE,FILE[ALL],FLD_HLP$,OSGUI$
```

Variable	Parameter	Description
HID\$	PGM(-2)	Program name
ROW	X0	Row of data field
COL	X1	Column of data field
PAGE	X2	Screen page number
FILE[ALL]	FILE[ALL]	The standard file channel array
FLD_HLP\$	FLD_HLP\$	Help screen ID
OSGUI\$	OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT3	GENGUHLP.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

The GENGUHLP.PUB program performs these functions:

- OPENS the help data file, xxHELP (where xx is the application ID)
- reads the help record, KEY=HID\$+PAGE+ROW+COL
- displays a window with a help record or a No Help Available message
- accepts a command character
- allows for an exit window, a delete help record, an add/change help record, and highlighting of key words

The routine traps errors and returns them to the CALLing program. The routine uses the SETESC verb to prohibit the user from pressing **Ctrl+Break** and exiting from the program. The access code for RMACHELP is checked before help can be edited or deleted. In edit mode GENGUHL.PUB uses GENINPUT. To highlight key words (display in foreground), place braces ({} around the words. When you exit from edit mode, the help message is redisplayed.

The PAGE variable allows screen locations that are used more than once in the same program. It should be initialized to 0 upon display of the first screen and adjusted as program logic dictates. Generally, this change takes place in the data prompt sections.

GENGUINP.PUB

GENGUINP.PUB is a public program for performing data entry. The interface routine GENINPUT should always be used to access GENGUINP.PUB.

```
0500 ENTER Z9$,Y$,ROW,COL,COM_COL,TYPE,LEN,FORCE,MASK$,X,X$,INFO,WINDID$,
0500:SELVAL$,COMPID$,PROGID$,FILE[ALL],YDISABLE$,HLP_ID$,FLD_HLP$,ID,
0500:BUTTON_KEYSS[ALL],OSGUI$,GUI_RADIO$,GENINPUT$
```

Variable	Parameter	Description
Z9\$	Z9\$	No function in GUI mode
Y\$	Y\$	Last command—set to the new command upon exit
ROW	(X0)	Row for data input
COL	(X1)	Column for data input
COM_COL	(X2)	Page reference for help
TYPE	(X3)	Type of input: 0 - string 1 - numeric 2 - string with a mask (such as a social security number) 3 - uppercase only string 5 - yes/no or toggled choice input with error trapping 6 - date field input with error trapping 7 - string (cursor is placed at end of field) 9 - other commands (single uppercase character) 10 - period input (blank entry not allowed) 11 - year input (blank entry not allowed) 12 - period/year input 13 - GL account number entry. Validates if GL Account is active 14 - Custom Edit Control entry (GUI only) 15 - toggled choice input (uses capitals in X1\$ for toggles) 17 - uppercase only string (cursor is placed at end of field)

Variable	Parameter	Description
TYPE (cont.)	(X3)	21 - period input (blank entry is allowed) 22 - year input (blank entry is allowed) 23 - period/year entry (blank entry is allowed) 24 - URL entry (Shift-F3 launches browser) 25 - email address entry (Shift-F3 launches email client) 26 - file browser dialog box (BBj only) 27 - directory browser dialog box (BBj only) 28 - file browser with path in STBL("BBJFILEPATH")(BBj only) 29 - GL account number entry. Does not validate if GL Account is active 30 - Word Wrap entry (used by GENADDL.PUB) 35 - Toggle, exit after toggle choice is made (GUI only) -8 - other commands (traps edit keys and function keys) -9 - other commands (traps edit keys also) -19 - other commands (traps edit and function keys; used only by GENGULIN.PUB)
LEN	(X4)	Length of input field
FORCE	X9	Forced entry flag: 0 - begin with default value in X or X\$ 1 - clear X or X\$ before data entry X9 is set to 0 upon exit.
MASK\$	X1\$(1)	Entry and display mask
X	X	Numeric default value AND return value
X\$	X\$	String default value AND return value
INFO	INFO	Information window status flag as defined in GENINPUT
WINDID\$	WINDID\$	Inquiry window ID as defined in GENINPUT
SELVAL\$	SELVAL\$	Inquiry selection value as defined in GENINPUT
COMPID\$	COMPID\$	Not currently used in GENINPUT.PUB

Variable	Parameter	Description
PROGID\$	PROGID\$	File maintenance program as defined in GENINPUT
FILE[ALL]	FILE[ALL]	Standard file array for inquiry and file maintenance
YDISABLE\$	YDISABLE\$	See the definition of YDISABLE\$ for GENINPUT
HLP_ID\$	PGM(-2)	The help ID for the program, usually the program name
FLD_HLP\$	FLD_HLP\$	Help ID of common-field help screen
ID	GUI_ID	Control ID waiting for input from user
BUTTON_KEYS\$ [ALL]	BUTTON_KEYS\$ [ALL]	Array to tie control ID's of buttons to character keystroke equivalents
OSGUI\$	OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more info)
GUI_RADIO\$	GUI_RADIO\$	String of radio buttons to parse though to control radio button selection
GENINPUT\$	GENINPUT\$	General Input Template

Exit Point	Explanation
EXIT	GENINPUT.PUB normally exits with no error.
EXIT ERR	An error is returned if an error occurred.
EXIT 127	An ESCAPE is returned if user breaks to BASIC (Ctrl+F10).

GENGUINP.PUB performs these functions:

- determines if a command character has been entered
- accepts the command sequence and returns the standard command
- processes the Esc and displays a command help window (GENINPUT.HLP)
- returns all other command characters to the CALLing program

- calls the GUI Inquiry Window routine when requested if WINDID\$ has value
- calls the Help routine when requested
- calls the Maintenance routine when requested if PROGID\$ has value
- allows Insert mode to be toggled on/off
- allows Quick mode to be toggled on/off
- allows verification to be toggled on/off
- allows the bell to be toggled on/off
- allows copying and pasting of data to and from controls
- displays online documentation
- validates periods and years for period/year entry
- displays Maint, Inquiry, Insert, Verify, Quick, Info and Company ID flags
- has an interface to run other commands or programs
- automatic error trapping when using X3=5 or X3=6
- restores the original data when the user presses the **UNDO** key
- performs debug functions as described below
- sets WINDID\$, SELVAL\$, and FLD_HLP\$ to nulls upon exiting

Multiple fields in the same function can share the same help record by setting the variable X2 to indicate the values of X0, X1 and X2 in the help file. The format is $X2=(X0*10000)+(X1*100)+X2$. Multiple fields in the same application can share the same help records by setting the variable FLD_HLP\$ with the help ID for the shared screen.

GENGULIB.PUB

GENGULIB.PUB is a public program that performs several functions, mainly for setting up menus, windows, and other information for the Graphical and Start-style menus. This program has seven entry points, which all use the same exit point.

MAKE_WINDOW_MENU:

This entry point **OPENS** the **sysgui** device “X0” and returns the channel on which it was **OPENED** in both the variable **WIND** and within **OSGUI.CHAN**. It also records the channel number into **STBL(“MGW”)**.

```
1000 MAKE_WINDOW_MENU:
1030 ENTER WIND,OSGUI$
```

Variable	Description
WIND	Channel on which the graphical device “X0” is open
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENGULIB.PUB does not normally return an error.
EXIT ERR	GENGULIB.PUB returns an error if one occurs.

NEW_WIND_MENU:

This entry point displays the Start-style menu and paints the image to be used with it. This procedure is **CALLED** only from **RMMENU1**.

```
1500 NEW_WIND_MENU:
1510 ENTER FILE[ALL],WIND,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array

Variable	Description
WIND	Channel on which the graphical device “X0” is open
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

MAKE_GUI_FILEBAR_MENU:

This entry point builds the standard menu bar that is used on all GUI menu types.

```
2000 MAKE_GUI_FILEBAR_MENU:
1510 ENTER WIND,OTHER$,RADIO_BUTTON$
```

Variable	Description
WIND	Channel on which the graphical device “X0” is open
OTHER\$	Template that contains the xxMN file structure. If OTHER\$ is left blank, the template will be retrieved from OSTPL.
RADIO_BUTTONS	If the value is non-zero, the Live or Sample radio button will be checked (on the Start-style menu only)

MAKE_OTHER_MENU:

This entry point builds the **Other Commands** menu and returns its structure to the CALLing program.

```
1505 MAKE_OTHER_MENU:
1510 ENTER MENU$,OTHER$,OVERRIDE_ID$
```

Variable	Description
MENU\$	Menu structure of the Other Commands menu
OTHER\$	Template that contains the xxMN file structure. If OTHER\$ is left blank, the template will be retrieved from OSTPL.
OVERRIDE_ID	Not used

POP_COMP_ID_MENU:

This entry point populates a combo box on the GUI menus with available company IDs and their names.

```
3510 POP_COMP_ID_MENU:
3520 ENTER WIND,OSCOMP,ID
```

Variable	Description
WIND	Channel on which the graphical device “X0” is open
OSCOMP	Channel on which the file OSCOMP is open
ID	Control ID to be populated (must be a List Button control)

GET_CURRENT_SCALE:

This entry point retrieves information about the Windows display, including screen width, height, and the scaling factor used when displaying OSAS screens. This factor is determined by dividing the display’s width and height by 640 and 480 respectively. This entry point is no longer used in standard OSAS programs.

```
3510 GET_CURRENT_SCALE:
3520 ENTER SINFO$
```

Variable	Description
SINFO	Template that returns information about the current display

This chart breaks down the SINFO\$ template:

Field	Description
GUICHAN:B	Channel on which the graphical device “X0” is open
SWIDTH:B	Current screen width
SHEIGHT:B	Current screen height
XSCALE:B	Current X scale, taking SINFO.SWIDTH/640
YSCALE:B	Current Y scale, taking SINFO.SHEIGHT/480

Field	Description
UNIT:C(1)	Current scaling units; 'P'ixels, 'S'emichars, or 'C'hars
FCSIZE:B	Font size in unscaled units
FSSIZE:B	Font size in scaled units
FSTYLE:B	Style of current font
FAMILY:C(1*)	Current font family

GET_CURRENT_FONT:

This entry point retrieves information about the current font used by the current window. The size of the font will always be 10.

```
5000 GET_CURRENT_FONT:
5010 ENTER SINFO$
```

Variable	Description
SINFO	Template that returns information about the current display

This chart breaks down the SINFO\$ template:

Field	Description
GUICHAN:B	Channel on which the graphical device "X0" is open
SWIDTH:B	Current screen width
SHEIGHT:B	Current screen height
XSCALE:B	Current X scale, taking SINFO.SWIDTH/640
YSCALE:B	Current Y scale, taking SINFO.SHEIGHT/480
UNIT:C(1)	Current scaling units; 'P'ixels, 'S'emichars, or 'C'hars
FCSIZE:B	Font size in unscaled units
FSSIZE:B	Font size in scaled units

Field	Description
FSTYLE:B	Style of current font, as represented by one or more of these values: \$01\$ = bold \$02\$ = italic \$10\$ = underline \$0200\$ = strikeout
FAMILY:C(1*)	Current font family

GENGULIN.PUB

GENGULIN.PUB is a public program that displays and processes commands for a GUI scrolling or *HLE* (horizontal line entry) screen.

```
0197 ENTER LN[ALL] , LOC[ALL] , LN$[ALL] , MASK$ , ACT$ , YDISABLE$ , TASK$ , GUI_LN[ALL] ,
0197 : ID , ROW_ID , BUTTON_KEYSS[ALL] , OSGUI$ , FLD_HLP$ , PASS_PROG$ , GENINPUT$
```

Variable	Description
LN[0]	-1 = Clear grid and exit 0 = do not display data 1 = display data 2 = display and exit, x+10 = display option, return any command keypress.
LN[1]	Not used
LN[2]	Total number of rows in the scroll region (a multiple of lines per item)
LN[3]	Lines per item
LN[4]	Starting column for scroll region
LN[5]	Starting row for scroll region
LN[6]	Maximum number of items in the scroll region (LN[2]/LN[3])
LN[7]	X value of the first LN\$[X,Y] cell displayed in the current scroll region
LN[8]	X value of the current LN\$[X,Y] cell (where the command prompt is never greater than LN[6])
LN[9]	Number of items in the current scroll region (never greater than LN[6])
LN[10]	Line number display column
LN[11]	Line number display row (if LN[11]=99, display line number for each item; otherwise, display only the current line number at column LN[10] and row LN[11])

Variable	Description
LN[12]	LN\$[X,Y] string truncate position (if zero, do not truncate). This allows data to be maintained at the end of the LN\$[X,Y] strings
LOC[0]	Maximum number of transaction lines allowed (for example, maximum of 998 lines in an order)
LOC[1]	First transaction line (sequence) displayed in the scroll region
LOC[2]	Current transaction line (where command prompt is)
LOC[3]	Total number of transaction lines (never more than LOC[0])
LOC[4]	"Info" indicator (0 = not available, 1 = available)
LOC[5]	Function level of screen colors (0, 1, 2, 3); 1 becomes 0
LN\$[X,Y]	Scroll region data (formatted strings)
MASK\$	Mask for line number display
ACT\$	Action returned to CALLing program (similar to Y\$ in GENINPUT)
YDISABLE\$	Reserved
TASK\$	Command returned to CALLing program
GUI_LN[X,1]	Stores ending position within LN\$[ALL] lines of each column
GUI_LN[X,2]	Alignment type: 0 - Left Alignment 1 - Right Alignment 2 - Centered 3 - Check box; a 'Y' in the first position in GUI_LN[X,1] determines whether the box is checked
ID	Control ID of the current grid
ROW_ID	Control ID of the current row display
BUTTON_KEY\$S [ALL]	Array to tie control ID's of buttons to character keystrokes
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)
FLD_HLP\$	Help ID of common-field help screen

Variable	Description
PASS_PROG\$	Name of program CALLing GENGULIN.PUB
GENINPUT\$	General Input template

LN[7] and LOC[1] are updated simultaneously, LN[8] and LOC[2] are updated simultaneously, and LN[9] and LOC[3] are updated simultaneously.

TASK\$ return values:

Control value	Meaning
LD	Line Down (down arrow and right arrow)
LU	Line Up (up arrow and left arrow)
PD	Page Down
PU	Page Up
EN	End
HO	Home
DL	Delete Line (Del)
IL	Insert Line (Ins)
TB	Tab Key

Valid one-character command include the numbers 0–9, the uppercase letters A–Z, and the symbols ?, <, >, @, *, +, -, =, and /.

GENGUOTH.PUB

GENGUOTH.PUB is a public program that CALLs a function that appears on the Other Commands menu within GUI screens. This function is CALLED only from GENGUINP.PUB.

```
0050 ENTER FILE[ALL],MENUITEM,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
MENUITEM	Control ID of the menu item selected
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENGUOTH.PUB does not return any errors.

GENGURES.PUB

GENGURES.PUB is a public program that loads all GUI screens from BRC files. This routine also builds standard tool bars and status bars on all screens as well.

```
0060 ENTER FILE[ALL],OSGUI$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	GUI template
MSG\$	Displays an error message if an error occurs

Exit Points	Explanation
EXIT	GENGURES.PUB usually does not return any errors.
EXIT ERR	GENGURES.PUB will return an error if one occurs.

This chart breaks down the OSGUI\$ template:

Field	Description
SCRNID\$:C(80*)	Name of .BRC file from which screens will be loaded
CHAN:N(5*)	Channels on which the graphical device “X0” is open
GUI_ID:N(5*)	Reserved for future development
LINE:N(5*)	Used by GENGULIN.PUB; returns the line of a grid that was clicked on
SCALE:N(5*)	Contains current scale factor (current window width/640)
CONTEXT_POINTER:N(5*)	Current context array
INQUIRY:N(1*)	Used by GENGUWIN.PUB; set to 1 so GENGUINP.PUB and GENGULIN.PUB know when an F2 window is active
RES_SCREEN:C(80*)	Not used

Field	Description
RES_HANDLE:N(3*)	Not used
CUR_CONTEXT[X]:N(5*)	Current context of the screen being loaded
BASE_CONTEXT[X]:N(5*)	Context of the last screen that has a status bar
TOOL_CONTEXT[X]:N(5*)	Context of the last screen that has a tool bar
TOOL_TYPE[X]:N(1*)	Set to 1 for 'Save' to appear in first text button on GUI screen, any other value will force 'OK' to appear. Currently this is not used.
WIND_ID[X]:N(1*)	Window ID from within the BRC file to load

GENGURES.PUB can open one form up at a time, or it can open several at once.

To load a single form, set OSGUI.SCRNID\$ to the name of the BRC you are working with, set OSGUI.WIND_ID[X] to the number of the window ID you want to load from the BRC file, and set OSGUI.CONTEXT_POINTER to whatever element of the array in OSGUI.WIND_ID[X] you are using.

To load multiple forms, set all of the window IDs into the OSGUI.WIND_ID[X] array. All forms must exist in the BRC file specified. Then, set OSGUI.CONTEXT_POINTER to the first array element you used to store the window IDs multiplied by -1. The negative value causes GENGURES.PUB to cycle through all window IDs you stored, loading each form. You can also set the first window ID negative in OSGUI.WIND_ID[X] to force GENGURES.PUB to display a speedo in that form's status bar, stating that other forms are loading, and to please wait.

If you have forms that either don't need a tool bar or don't have a status bar, you can set OSGUI.TOOL_CONTEXT[X] to -1 to stop a tool bar from being displayed, and OSGUI.BASE_CONTEXT[X] to -1 to stop a status bar from being displayed. However, after a form loads, you must set OSGUI.BASE_CONTEXT[X] to point to a context where there is a status bar.

GENGUUSR.PUB

GENGUUSR.PUB is a public program that CALLs a function that appears on the F4 User Commands menu within GUI menus. This function is CALLED only from RMMENU2 and RMMDIMNU.

```
0050 ENTER FILE[ALL],MENUITEM,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
MENUITEM	Control ID of the menu item selected
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENGUUSR.PUB does not return any errors.

GENGUSER.PUB

GENGUSER.PUB is a public program that retrieves a user ID and description from the OSUS file, given an associated terminal ID. If a matching entry is not found for the given terminal ID in the OSUS file, the description from the OSDF file is returned instead.

```
0050 ENTER TERM$,USERID$,USERDESC$
```

Variable	Description
TERM\$	Terminal ID
USERID	The User ID from the OSUS file is returned.
USERDESC\$	The User Description is returned from the OSUS file, or from the OSDF file if not found in OSUS.

Exit Points	Explanation
EXIT	GENGUSER.PUB does not return any errors.

GENGUWIN.PUB

GENGUWIN.PUB is a public program that displays inquiry windows in GUI mode. This program is CALLED only by GENGUINP.PUB.

```
0100 ENTER Z9$,WINDID$,RETVAL$,SELVAL$,COMPID$,FILE[ALL],OSGUI$
```

Variable	Description
Z9\$	Unused
WINDID\$	Window ID of the inquiry window record in the xxWIND file
RETVAL\$	Starting value for entries in the window—set to the selected value when GENGUWIN.PUB exits
SELVAL\$	Selection value for limiting the entries that appear in the window
COMPID\$	Unused
FILE[ALL]	Standard file array used for the data file channel number
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENGUWIN.PUB normally exits with no error.
EXIT ERR	An error is returned if an error occurs.

For a detailed explanation on how GENGUWIN.PUB works, see the section on GENWIND.PUB.

GENHELP.PUB

GENHELP.PUB is a public program for displaying context-sensitive help in text mode. This program is intended for GENINPUT.PUB and is invoked when the **Help** key is pressed. All parameters originate from GENINPUT.

```
0100 ENTER HID$,ROW,COL,PAGE,FILE[ALL],FLD_HLP$
```

Variable	Parameter	Description
HID\$	PGM(-2)	Program name
ROW	X0	Row of data field
COL	X1	Column of data field
PAGE	X2	Screen page number
FILE[ALL]	FILE[ALL]	The standard file channel array
FLD_HLP\$	FLD_HLP\$	Help screen ID

Exit Points	Explanation
EXIT	GENHELP.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

The GENHELP.PUB program performs these functions:

- OPENS the help data file, xxHELP (where xx is the application ID)
- reads the help record, KEY=HID\$+PAGE+ROW+COL
- displays a window with a help record or a No Help Available message
- accepts a command character
- allows for an exit window, a delete help record, an add/change help record, and highlighting of key words

The routine traps errors and returns them to the calling program. The routine uses the SETESC verb to prohibit the user from pressing **Ctrl+Break** and exiting from the program. The access code for RMACHELP is checked before help can be edited or deleted. In edit mode GENHELP.PUB uses GENINPUT. To highlight key words (display in foreground), place braces ({}) around the words. When you exit from edit mode, the help message is redisplayed.

The PAGE variable allows screen locations that are used more than once in the same program. It should be initialized to 0 upon display of the first screen and adjusted as program logic dictates. Generally, this change takes place in the data prompt sections.

GENINFO (INFO)

GENINFO (INFO) is a general routine that calls GENINFO.PUB to display and process an information window menu.

Variable	Description
APPKEY\$	List of valid applications to process
KEYNAME\$[ALL]	Default keys for each valid application
FILE[ALL]	Standard file array for inquiry and file maintenance
O2\$	Options and interfaces string
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

The string APPKEY\$ contains a list of two-character application IDs that are valid for display on the information menu. An example value for APPKEY\$ is "ARIN". This means that the applications AR (Accounts Receivable) and IN (Inventory) are valid to be displayed. The variable KEYNAME\$[1] would then contain the default customer ID, and the variable KEYNAME\$[2] would contain the default item ID.

GENINFO.PUB

GENINFO.PUB is a public program used for displaying and processing an applications information window menu in text mode.

```
0090 ENTER APPKEY$,KEYNAME$[ALL],FILE[ALL],O2,OSGUI$
```

Variable	Description
APPKEY\$	List of valid applications to process
KEYNAME\$[ALL]	Default keys for each valid application
FILE[ALL]	Standard file array for inquiry and file maintenance
O2\$	Options and interfaces string
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more detail)

Exit Points	Explanation
EXIT	GENINFO.PUB normally exits with no error.

GENINFO.PUB reads the record xxINFO in the file xxMNFL. The menu is then displayed, and keyboard commands are processed by a call to the public program RMINFO.PUB.

GENINPUT

GENINPUT is a general routine that interfaces to GENINPUT.PUB and GENGUINP.PUB for data entry. It occupies lines 6000–6199 and should be merged into all interactive programs. GENINPUT requires that GENERERROR also be merged into the program.

Variable	Description
X0	Row for data input
X1	Column for data input
X2	Page index for context-sensitive help
X3	Type of input: 0 - string 1 - numeric 2 - string with a mask (such as a social security number) 3 - uppercase only string 5 - yes/no or toggled choice input with error trapping 6 - date field input with error trapping 7 - string (cursor is placed at end of field) 9 - other commands (single uppercase character) 10 - period input (blank entry not allowed) 11 - year input (blank entry not allowed) 12 - period/year input 13 - GL account number entry. Validates if GL Account is active 14 - Custom Edit Control entry (GUI only) 15 - toggled choice input (uses capitals in X1\$ for toggles) 17 - uppercase only string (cursor is placed at end of field) 21 - period input (blank entry is allowed) 22 - year input (blank entry is allowed) 23 - period/year entry (blank entry is allowed) 24 - URL entry (Shift-F3 launches browser) 25 - email address entry (Shift-F3 launches email client) 26 - file browser dialog box (BBj only) 27 - directory browser dialog box (BBj only) 28 - file browser with path in STBL("BBJFILEPATH") (BBj only) 29 - GL account number entry. Does not validate if GL Account is active

Variable	Description
X3 (continued)	30 - Word Wrap entry (used by GENADDL.PUB) 35 - Toggle, exit after toggle choice is made (GUI only) -8 - other commands (traps edit keys and function keys) -9 - other commands (traps edit keys also)
X4	Length of input field
X1\$	Entry and display mask for numeric and string with a mask input
X9	Forced entry flag: 0 - begin with default value in X or X\$ 1 - clear X or X\$ before data entry X9 is set to 0 upon exit
X\$	String default value—set to value entered by user for string input
X	Numeric default value—set to value entered by user for numeric input
Y\$	Last command—a value of "P" or "J" and X9 set to 0 will cause GENINPUT to return without calling GENINPUT.PUB or GENGUINP.PUB for input. Y\$ is set by GENINPUT.PUB or GENGUINP.PUB to the appropriate command value based on the last key the user pressed.
INFO	Information window status flag (0 = disabled, 1 = display INFO, 2 = display DETL)
WINDID\$	Inquiry window ID for the data field. The Inquiry key is available if WINDID\$ contains a value. WINDID\$ is cleared before GENINPUT returns.
SELVAL\$	Selection value for the inquiry window. This value, coupled with the inclusion and exclusion masks defined in the window, can be used to limit the entries that appear in an inquiry window. SELVAL\$ is always cleared before GENINPUT returns.
PROGID\$	File maintenance program for the data field. The Maintenance key is available if PROGID\$ contains a value.

Variable	Description
FILE[ALL]	Standard file array for inquiry and file maintenance
YDISABLE\$	String of Y\$ values that should be ignored. Only ESC sequence commands can be disabled (that is, Esc+X for Abandon, but no Ctrl+K for Up).
PGM(-2)	The standard BBx variable containing the name of the CALLing program
FLD_HLP\$	Help ID of common-field help screen
GUI_ID	GUI control ID of current entry field
BUTTON_KEY\$S [ALL]	Array to tie control ID's of buttons to character keystroke equivalents
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57)
GUI_RADIO\$	String of radio buttons to parse though when controlling radio button selection
GENINPUT\$	General Input Template

Exit Points	Explanation
RETURN	GENINPUT normally RETURNs.
GOSUB GENERERROR	GENERERROR is called to process special commands in Y\$. This makes GENINPUT subject to the exit points defined in GENERERROR.
GOSUB 9200	The delete record section is called if Y\$ is "D". Once verified, this section exits by simulating the pressing of the Abandon key in GENERERROR.

This chart breaks down the GENINPUT\$ template:

Field	Description
DRILL:N(1*)	Set to 1 if current field/HLE line is drillable
COL:N(2*)	Column of HLE that was double clicked
ROW:N(2*)	Row of HLE that was double clicked

GENINPUT.PUB

GENINPUT.PUB is a public program for performing data entry. The interface routine GENINPUT should always be used to access GENINPUT.PUB.

```
0500 ENTER Z9$,Y$,ROW,COL,COM_COL,TYPE,LEN,FORCE,MASK$,X,X$,INFO,WINDID$,
0500:SELVAL$,COMPID$,PROGID$,FILE[ALL],YDISABLE$,HLP_ID$,FLD_HLP$,GENINPUT$
```

Variable	Parameter	Description
Z9\$	Z9\$	Alternate field highlight. Set to brackets for terminals with no reverse video. Cleared for all other terminals.
Y\$	Y\$	Last command—set to new command on exit
ROW	(X0)	Row for data input
COL	(X1)	Column for data input
COM_COL	(X2)	Page reference for help
TYPE	(X3)	Type of input: 0 - string 1 - numeric 2 - string with a mask 3 - string forced to uppercase 5 - yes/no or choice field entry 6 - date field entry 7 - string (cursor at end of field) 9 - other commands (one uppercase character) 10 - period entry 11 - year entry 12 - period/year entry 13 - GL account number entry. Validates if GL Account is active 14 - Custom Edit Control entry (GUI only) 15 - toggled input (use capitals in X1\$ for toggles) 17 - uppercase only string (cursor at end of field) 21 - period input (blank entry is allowed) 22 - year input (blank entry is allowed) 23 - period/year entry (blank entry is allowed) 24 - URL entry (Shift-F3 launches browser) 25 - email address (Shift-F3 launches email client) 26 - file browser dialog box (BBj only) 27 - directory browser dialog box (BBj only) 28 - file browser with path in STBL("BBJFILEPATH") (BBj only)

Variable	Parameter	Description
TYPE (cont.)	(X3)	29 - GL account number entry. Does not validate if GL Account is active 30 - Word Wrap entry (used by GENADDL.PUB) 35 - Toggle, exit after toggle choice is made (GUI only) -8 - other commands (traps edit and function keys) -9 - other commands (traps edit keys also)
LEN	(X4)	Length of input field
FORCE	X9	Forced entry flag: 0 - begin with default value in X or X\$ 1 - clear X or X\$ before data entry X9 is set to 0 upon exit
MASK\$	X1\$(1)	Entry and display mask
X	X	Numeric default value AND return value
X\$	X\$	String default value AND return value
INFO	INFO	Information window status flag as defined in GENINPUT
WINDID\$	WINDID\$	Inquiry window ID as defined in GENINPUT
SELVAL\$	SELVAL\$	Inquiry selection value as defined in GENINPUT
COMPID\$	COMPID\$	Not currently used in GENINPUT.PUB
PROGID\$	PROGID\$	File maintenance program as defined in GENINPUT
FILE[ALL]	FILE[ALL]	Standard file array for inquiry and file maintenance
YDISABLE\$	YDISABLE\$	See the definition of YDISABLE\$ for GENINPUT
HLP_ID\$	PGM(-2)	The help ID for the program, usually the program name
FLD_HLP\$	FLD_HLP\$	Help ID of common-field help screen

Exit Point	Explanation
EXIT	GENINPUT.PUB normally exits with no error.
EXIT ERR	An error is returned if an error occurred.
EXIT 127	An ESCAPE is returned if Ctrl+A is pressed in debug mode. Debug mode is toggled with an ESCAPE (Ctrl+Break on PCs).

This chart breaks down the GENINPUT\$ template:

Field	Description
DRILL:N(1*)	Set to 1 if current field/HLE line is drillable
COL:N(2*)	Column of HLE that was double clicked
ROW:N(2*)	Row of HLE that was double clicked

GENINPUT.PUB performs these functions:

- determines if a command character has been entered
- accepts the command sequence and returns the standard command
- processes the **Esc** and displays a command help window (GENINPUT.HLP)
- returns all other command characters to the CALLing program
- calls the Inquiry Window routine when requested if WINDID\$ has value
- calls the Help routine when requested
- calls the Maintenance routine when requested if PROGID\$ has value
- allows Insert mode to be toggled on/off
- allows Quick mode to be toggled on/off
- allows verification to be toggled on/off
- allows the bell to be toggled on/off

- allows copying and pasting of data to and from fields
- displays online documentation
- validates periods and years for period/year entry
- displays Maint, Inquiry, Insert, Verify, Quick, Info and Company ID flags
- has an interface to run other commands or programs
- accepts user input a single character at a time
- automatic error trapping when using X3=5 or X3=6
- warns the user with a beep when he enters an invalid character
- warns the user with a beep when he types too many characters
- restores the original data when the user presses the **UNDO** key
- performs debug functions as described below
- sets WINDID\$ and SELVAL\$ to nulls upon exiting

Multiple fields in the same function can share the same help record by setting the variable X2 to indicate the values of X0, X1 and X2 in the help file. The format is $X2 = (X0 * 10000) + (X1 * 100) + X2$. Multiple fields in the same application can share the same help records by setting the variable FLD_HLP\$ with the help ID for the shared screen.

The user can toggle debug mode by pressing **Ctrl+C (Ctrl+F10)** in graphical mode). In debug mode the user can enter additional commands that perform debugging functions. These three functions are available in debug mode:

- Control+A (break), exits and returns an error 127 to the calling program.
- Control+S, dump to screen, performs the DUMP verb to the screen.
- Control+P, dump to printer, performs the DUMP verb to the LP printer.

GENLABEL.PUB

GENLABEL.PUB is a public program that prints labels for many key files, such as employees, customers, and vendors. Label setup is maintained thru Resource Manager, and then selected as a base for labels in OSAS applications.

SETUP:

This entry point sets up the SCRT\$ template used to hold information about the scratch file that contains the data to print on labels.

```
1005 SETUP:
1010 ENTER FILE[ALL] , OSLB$ , SCRT$
```

Variable	Description
FILE[ALL]	Standard file array
OSLB\$	Current label template from the label ID the user selects to print their labels
SCRT\$	Template for scratch file used to hold data that prints on labels

This chart breaks down the SCRT\$ template:

Field	Description
SEQ:C(8*)	Unique sequence number
LINES[OSLB.HEIGHT]: C(OSLB.WIDTH)	Array of lines that holds the data that prints on label. The array is dimesioned to however many lines there are per label, by number of characters wide each label is

START_PRINT:

This entry point is used when the calling program is ready to start printing the data that's stored in the scratch file built by the calling program, in the format laid out by the SCRT\$ template.

```
2005 START_PRINT:
2010 ENTER FILE[ALL] , OSLB$, SCRT_INDEX, SCRT$, PRINT_CHANNEL, COPIES, COLLATE
```

Variable	Description
FILE[ALL]	Standard file array
OSLB\$	Current label template from the label ID the user selects to print their labels
SCRT_INDEX	Index to FILE[ALL] element where the scratch file that holds the data to print is opened on
SCRT\$	Template for scratch file used to hold data that prints on labels
PRINT_CHANNEL	Channel where printer or print file is opened on
COPIES	Number of copies to print for each label
COLLATE	0=Normal printing, 1=Collated printing

GENLINE

GENLINE is a general routine that interfaces to GENLINE.PUB and GENGULIN.PUB for displaying and processing commands for a scrolling horizontal line entry screen. GENLINE occupies lines 6700–6780 and should be merged into all programs using horizontal line entry.

Variable	Description
LN[ALL]	Display variables
LOC[ALL]	Local program variables
LN\$[ALL]	Formatted scroll region data
MASK\$	Mask for line number display
ACT\$	Action returned (similar to Y\$ in GENINPUT)
DISABLE\$	Reserved
TASK\$	Command returned
GUI_LN[ALL]	Column information on how LN\$[ALL] breaks down
GUI_ID	Control ID of grid
ROW_ID	Control ID of the current row display
BUTTON_KEY\$[ALL]	Control ID's of command buttons and their keystroke equivalent
OSGUI\$	GUI template (see GENGURES.PUB for more detail on page 3-57)
FLD_HLP\$	Help ID of common-field help screen
PGM(-2)	Name of current program
GENINPUT\$	General Input template

This chart breaks down the GENINPUT\$ template:

Field	Description
DRILL:N(1*)	Set to 1 if current field/HLE line is drillable
COL:N(2*)	Column of HLE that was double clicked
ROW:N(2*)	Row of HLE that was double clicked

GENLINE.PUB

GENLINE.PUB is a public program that displays and processes commands for a scrolling horizontal line entry screen.

```
0200 ENTER LN[ALL], LOC[ALL], LN$[ALL], MASK$, ACT$, YDISABLE$, TASK$, FLD_HLP$,
0200: PASS_PROG$, GENINPUT$
```

Variable	Description
LN[0]	0 = do not display data 1 = display data 2 = display and exit x+10 = display option, return any command keypress
LN[1]	Column for command prompt (>)
LN[2]	Total number of rows in the scroll region (a multiple of lines per item)
LN[3]	Lines per item
LN[4]	Starting column for scroll region
LN[5]	Starting row for scroll region
LN[6]	Maximum number of items in the scroll region (LN[2]/LN[3])
LN[7]	X value of the first LN\$[X,Y] cell displayed in the current scroll region
LN[8]	X value of the current LN\$[X,Y] cell (where the command prompt is never greater than LN[6])
LN[9]	Number of items in the current scroll region (never greater than LN[6])
LN[10]	Line number display column
LN[11]	Line number display row (if LN[11]=99, display line number for each item; otherwise, display only the current line number at column LN[10] and row LN[11])
LN[12]	LN\$[X,Y] string truncate position (if zero, do not truncate)—this allows data to be maintained at the end of the LN\$[X,Y] strings

Variable	Description
LOC[0]	Maximum number of transaction lines allowed (for example, maximum of 998 lines in an order)
LOC[1]	First transaction line (sequence) displayed in the current scroll region
LOC[2]	Current transaction line (where command prompt is)
LOC[3]	Total number of transaction lines in use (never more than LOC[0])
LOC[4]	"Info" indicator (0 = not available, 1 = available)
LOC[5]	Function level of screen colors (0, 1, 2, 3); 1 becomes 0
LN[X,Y]	Scroll region data (formatted strings)
MASK\$	Mask for line number display
ACT\$	Action returned to CALLing program (similar to Y\$ in GENINPUT)
YDISABLE\$	Reserved
TASK\$	Command returned to CALLing program
FLD_HLP\$	Help ID of common-field help screen
PASS_PROG\$	Name of program calling GENLINE.PUB
GENINPUT\$	General Input template

LN[7] and LOC[1] are updated simultaneously, LN[8] and LOC[2] are updated simultaneously, and LN[9] and LOC[3] are updated simultaneously.

TASK\$ return values:

Control value	Meaning
LD	Line Down (down arrow and right arrow)
LU	Line Up (up arrow and left arrow)
PD	Page Down
PU	Page Up
EN	End

Control value	Meaning
HO	Home
DL	Delete Line (Del)
IL	Insert Line (Ins)
TB	Tab Key

Valid one-character command include the numbers 0–9, the uppercase letters A–Z, and the symbols ?, <, >, @, *, +, -, =, and /.

This chart breaks down the GENINPUT\$ template:

Field	Description
DRILL:N(1*)	Set to 1 if current field/HLE line is drillable
COL:N(2*)	Column of HLE that was double clicked
ROW:N(2*)	Row of HLE that was double clicked

GENLIST

GENLIST is a general routine that contains a multiline function called FNLIST. This function takes a control ID, a string, a list of possible values, and a string length, and based on that information, will select a value from a list box based on the information. GENLIST occupies lines 18900–18950 and should be merged into all programs that have list buttons on a GUI screen.

Variable	Description
ID	Control ID of list button
PARAM\$	String to display on screen
LIST\$	List of all strings to populate button
LEN	Length of each string, equal to X4

To use GENLIST, add a line in the GUI_DATA section where a selection value from a toggle option would normally print in the DATA section of code. For example, if the code in the data block reads like this:

```
7750 PRINT @(47,12),CALCFIN$
```

its GUI counterpart would read as follows:

```
17750 LET TMP=FNLIST(4030,CALCFIN$,"Invoice DateDue Date",12)
```

Notice that you must pass all possible options for the toggle and the length of each option in order for GENLIST to determine where the value comes from. This is necessary in order to make the proper selection in the GUI list button.

GENLOG.PUB

GENLGO.PUB is a public program that performs several functions related to user logging. It's key function is to write new entries to the user log (OSLG) file.

TEMPLATE_INIT:

This entry point sets up the OSLG template, which is used throughout the user log processes.

```
1005 TEMPLATE_INIT:
1001 ENTER OSLG$
```

Variable	Description
OSLG\$	Template for data file OSLG

Exit Points	Explanation
EXIT	GENCVTTP.PUB does not normally return an error.

This chart breaks down the OSLG\$ template. All values are set by the WRITE_NEW_LOG call point, unless noted that the variable can be set by the calling program:

Field	Description
USRID:C(12*)	STBL("USERID")
SYSDATE:N(7*)	Current system date
TIME:C(8*)	Current system time
TERMID:C(4*)	Current terminal ID
APP:C(2*)	STBL("MENUAPP"); May be set by calling program
MENU:C(40*)	STBL("MENUDESC"); May be set by calling program
LIVE:N(1*)	0=Sample data, 1=Live data
CID:C(3*)	STBL("CID"); May be set by calling program
WKSDATE:N(7*)	STBL("JULDATE") (current workstation date)
CODECOMP:C(3*)	STBL("CID")

Field	Description
CODECODE:C(12*)	STBL("ACCESSCODE")
CODENAME:C(12*)	STBL("ACCESSNAME")
CODEMENU:C(1*)	STBL("MITEM"); May be set by calling program
MENUNAME:C(50*)	STBL("MPROG"); May be set by calling program
CODEPARM:C(2*)	STBL("MPARAM"); May be set by calling program
MENUREC:C(12*)	Not used
MENUFILE:C(12*)	Not used
EXITDATE:N(7*)	Sets to system date when exiting a function
EXITTIME:C(8*)	Sets to system time when exiting a function
APPYEAR:N(4*)	If calling program is PA/DD/GL, sets to current year of that application
RSV1:C(1*)	Reserved for OSD
RSV2:C(1*)	Reserved for OSD
RSV3:N(1*)	Reserved for OSD
RSV4:N(1*)	Reserved for OSD
ISVA:C(1*)	Reserved for ISV
ISVN[2]:N(1*)	Reserved for ISV

WRITE_NEW_LOG:

This entry point is used to write new entries to the log file. It is called by the menu programs before the user is taken to the function they chose to run. A global variable is set that contains the key to the record so that the exit time can be recorded once the user leaves the function. This is kept in STBL("LOGENTRY").

```
2005 WRITE_NEW_LOG:
2010 ENTER OSLG$,FILE[ALL]
```

Variable	Description
OSLG\$	Template for data file OSLG
FILE[ALL]	Standard file array

WRITE_TIMEOUT_LOG:

This entry point is used to write the current system date and time to the exit fields in the OSLG file, given a specific key in the file contained in STBL("LOGENTRY").

```
2205 WRITE_TIMEOUT_LOG:
2210 ENTER OSLG$, FILE [ALL]
```

Variable	Description
OSLG\$	Template for data file OSLG
FILE[ALL]	Standard file array

STARTING_STOPPING_OSAS:

This entry point is used to record when a user started and exited OSAS. The starting entry record is saved in STBL("LOGENTRYSTARTOSAS"), and if set, when this function is called, it's assuming that the user is exiting OSAS.

```
2305 STARTING_STOPPING_OSAS:
2310 ENTER OSLG$, FILE [ALL]
```

Variable	Description
OSLG\$	Template for data file OSLG
FILE[ALL]	Standard file array

GENLOCK

GENLOCK is general routine that interfaces to GENERERROR.PUB for handling program errors and ESCAPEs. GENLOCK occupies lines 9300–9399 and should be merged into all non interactive programs. The description of GENERERROR details all variables that are passed to GENERERROR.PUB. Below are the variables that are relevant to GENLOCK:

Variable	Description
O7	GENLOCK sets O7 to -1 to indicate that an automatic retry should occur if the current error warrants a retry attempt. O7 is set to -2 to indicate that an Escape was detected. O7 is set to 0 before GENERERROR.PUB exits.
Y\$	Y\$ is set to the appropriate value after a key is pressed before GENERERROR.PUB exits. For an automatic retry, Y\$ is set to "R", the message is displayed, and there is a slight pause until GENERERROR.PUB exits.
X1\$	Mask that caused a mask format error
F\$	File name that caused a file error
YDISABLE\$	String of Y\$ values that should be ignored. Note that only ESC sequence commands can be disabled (that is, Esc+X for Abandon, but not Ctrl+K for Up)
OSGUI\$	GUI template (see GENGURES.PUB for more detail)

Exit Points	Explanation
RETURN	GENLOCK returns after an Escape is detected, but the user elects to continue processing.
GOTO 9990	Y\$ = "M" for an Exit key press (and verified if necessary).
RETRY	A RETRY is attempted for all other values of Y\$.

GENMAINT.PUB

GENMAINT.PUB is a public program that interfaces between GENINPUT.PUB and a file maintenance program when the **Maintenance** key is pressed. GENMAINT.PUB is intended for use by GENINPUT.PUB only.

Variable	Description
FILE[ALL]	Standard file array
EDIT_DEF\$	Current data field value. The file maintenance program should set this field to the value of the last record saved.
PROGID\$	Name of the file maintenance program to call
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more details)

Exit Points	Explanation
EXIT	GENMAINT.PUB never returns an error.

The application owning the file maintenance program is determined from the first two characters of PROGID\$. GENMAINT.PUB switches to the application directory, loads the application option switches, creates a new program window and calls the file maintenance program in PROGID\$ as follows:

```
1150 CALL PROGID$,FIL[ALL],EDIT_DEF$,O2$
```

Variable	Description
FILE[ALL]	Standard file array
EDIT_DEF\$	Current data field value. The file maintenance program should set this field to the value of the last record saved.
O2\$	Option switch settings for application specified by the first two characters of PROGID\$

When control is returned to GENMAINT.PUB, the program window pops up, the current directory is restored to the original application directory, and control is returned to GENINPUT.PUB. Any additional variables that are needed from the calling application program to PROGID\$ must be set in the calling application program as global variables at the time PROGID\$ is set. Global variables and PROGID\$ should be set only if the application being referenced exists and the application being referenced is interfaced with the current application (cross applications only).

GENMPATH.PUB

GENMPATH.PUB is a public program that performs several functions, all related to determining and changing to application modification directories.

GET_SET_MOD_PATH:

This entry point will get an applications modifications directory and change to it, adding the base application directory to the prefix.

```
0050 GET_SET_MOD_PATH:
0060 ENTER APP$,MSG$,CH_DIR
```

Variable	Description
APP\$	Application ID
MSG\$	Error message returned to calling program
CH_DIR	Internal use only

Exit Points	Explanation
EXIT	GENMPATH.PUB does not normally return an error.
EXIT ERR	GENMPATH.PUB returns untrapped errors if one occurs.

BUILD_COMP_MOD_PATH:

This entry point will take a specified path and attempt to find a company specific path on it, if the letters 'xxx' exist in the path. If variable will return the correct path if one is found, or blank if no directory is found.

```
3005 BUILD_COMP_MOD_PATH:
3010 ENTER PATH$
```

Variable	Description
PATH\$	Directory to evaluate

Exit Points	Explanation
EXIT	GENMPATH.PUB does not normally return an error.
EXIT ERR	GENMPATH.PUB returns untrapped errors if one occurs.

GENNUM

GENNUM is a general routine that contains two multiline functions called FNNUM\$ and FNGUINUM\$. These functions take a numeric value and a mask and return a string containing the masked numeric value. FNGUINUM\$ also sets a mask for an INPUTN control. GENNUM occupies lines 10000–10099 and should be merged into all programs that display or print numeric values.

Variable	Description
XX3	Any numeric value to format
XX1\$	The numeric mask to use
ID (FNGUINUM\$ only)	A GUI control ID in which the mask and data should be set

To use GENNUM, substitute FNNUM\$(XX,XX1\$) for the old XX:XX1\$ logic in screen display or report printing. For example, to print the variable A6 to the screen using the mask in M1\$,

```
7720 PRINT @(22,10),FNNUM$(A6,M1$),
```

instead of

```
7720 PRINT @(22,10),A6:M1$,
```

In GUI mode, the above code would look like the following:

```
17720 PRINT (OSGUI.CHAN)'TITLE'(4010,FNGUINUM$(A6,M1$,4010)
```

This would set the mask M1\$ as the current mask for control ID 4010, and also display the data to the control.

If the value of A6 is too large to fit the mask, the function returns asterisks (*) to the length of the mask, providing protection against error 43s. For GUI screens, asterisks will not be displayed. Error 43s will still be protected against in GUI mode, but the control will be filled with spaces rather than asterisks.

GENOPEN

GENOPEN is a general routine that interfaces to GENOPEN.PUB to OPEN and CLOSE data files. GENOPEN occupies lines 8300–8399 and should be merged into all programs that OPEN data files. This chart breaks down the OSGUI\$ This

Variable	Description
F0	File array index to store channel number. Valid numbers are 1–70 [actually 1 to NUM(STBL("CHANNELS"))]. Use a negative array index to indicate that the file should be locked.
F\$	The name of the file to open
FILE[F0]	The channel of the OPENed file is stored in the file array
TASK	Determines the task GENOPEN.PUB is to perform. 0 = OPEN the file in F\$ on the index array channel in F0 1 = Get the string template for the file in F\$ 2 = OPEN the file in F\$ on the index array channel in F0 and get the file's string template 3 = Search for the file in F\$ in the FILE[] array and return the channel index (if found) in F0 4 = CLOSE the data file on index array channel F0 5 = Search for the file in F\$ in the FILE[] array and return the channel index (if found) in F0, or open the file if the file was not found and return the FILE[] array index in F0 (if F0 is negative on return, the file was opened in GENOPEN.PUB) 6 = Search for the file in F\$ in the FILE[] array and return the channel index (if found) in F0, or open the file if the file was not found and return the FILE[] array index in F0 (if F0 is negative on return, the file was opened in GENOPEN.PUB), and return the file's string template

Variable	Description
TASK (continued)	7 = Search the FILE[] array for the first zero element, and open the file on that element, returning the element position in F0 8 = Search the FILE[] array for the first zero element, open the file on that element, returning the element position in F0, and return the string template of the file 9 = Check if the file in F\$ exists, returning TASK=TASK*-1 if file is not found 10 = Unlock the file on FILE[F0], returning GENOPEN.RET="U" and error message in MSG\$ if an ERR=14 is encountered 11 = Locks the file on FILE[F0], returning GENOPEN.RET="U" and error message in MSG\$ if any error is encountered.
TPL\$	GENOPEN.PUB returns the string template retrieved (if any)
GENOPEN\$	General Open Template

Exit Points	Explanation
RETURN	GENOPEN returns after opening the file.
ERR=GENERERROR	Errors branch to GENERERROR. If possible, the user can retry the open; otherwise, he is forced to exit.

This chart breaks down the GENOPEN\$ template:

Field	Description
CRYP\$:N(1*)	Set to 1 if file to be opened is encrypted
RET:C(1*)	Set to "U" by GENOPEN.PUB when file is unavailable to be locked or unlocked

GENOPEN.PUB

GENOPEN.PUB is a public program that OPENS and CLOSEs data files.

```
0100 ENTER FILE[ALL], FILE$, IND, TASK, TPL$, MSG$, Y$, GENOPEN$
```

Variable	Description
FILE[ALL]	Standard file array
FILE\$	File to open
IND	FILE[] array index. Values are 1- NUM(STBL("CHANNELS")). A negative array index indicates that the file should be locked.
TASK	Determines the task GENOPEN.PUB is to perform. 0 = OPEN the file in F\$ on the index array channel in F0 1 = Get the string template for the file in F\$ 2 = OPEN the file in F\$ on the index array channel in F0 and get the file's string template 3 = Search for the file in F\$ in the FILE[] array and return the channel index (if found) in F0 4 = CLOSE the data file on index array channel F0 5 = Search for the file in F\$ in the FILE[] array and return the channel index (if found) in F0, or open the file if the file was not found and return the FILE[] array index in F0 (if TASK is negative on return, the file was OPENed by GENOPEN.PUB) 6 = Search for the file in F\$ in the FILE[] array and return the channel index (if found) in F0, or open the file and retrieve the string template if the file was not found

Variable	Description
TASK (continued)	7 = Search the FILE[] array for the first zero element, and open the file on that element, returning the element position in F0 8 = Search the FILE[] array for the first zero element, open the file on that element, returning the element position in F0, and return the string template of the file 9 = Check if the file in F\$ exists, returning TASK=TASK*-1 if file is not found 10 = Unlock the file on FILE[F0], returning GENOPEN.RET="U" and error message in MSG\$ if an ERR=14 is encountered 11 = Locks the file on FILE[F0], returning GENOPEN.RET="U" and error message in MSG\$ if any error is encountered.
TPL\$	GENOPEN.PUB returns the string template retrieved (if any)
MSG\$	An error message is returned if the operation is unsuccessful
Y\$	GENOPEN.PUB and the CALLing program share the current command, in case the user aborts processing
GENOPEN\$	General Open Template

Exit Points	Explanation
EXIT	GENOPEN.PUB never returns an error. If a file OPEN is unsuccessful, a message is returned in MSG\$.

This chart breaks down the GENOPEN\$ template:

Field	Description
CRYP\$:N(1*)	Set to 1 if file to be opened is encrypted
RET:C(1*)	Set to "U" by GENOPEN.PUB when file is unavailable to be locked or unlocked

GENOTBBJ.PUB

GENOTBBJ.PUB is a public program for selecting the output destination when printing to a BBj PDF file or print object. This program is currently called from customer invoices and statements, vendor vouchers and purchase orders, and employee vouchers.

```
0110 ENTER DEVICE$,CHANNEL,OUT_COLS,BBJPRINTER$,BBJPATH$,FORM_TYPE$,COPIES,
PRINT_DEVICE$,PRINT_CHANNEL,LINES,COPY_NAME$[ALL],ALIGNMENT
```

Variable	Description
DEVICE\$	Devices that can be selected. One or more of the following options are valid: E = all P = printer F = file S = Screen R = print preview W = Lotus worksheet output DEVICE\$ is set to the alias of the OPENed device upon exit.
CHANNEL	Set to the channel number of the device. If the channel comes in as 123, a WKS file type is added to the list of devices for Lotus output.
OUT_COLS	Number of output columns, usually 80 or 132. OUT_COLS can be changed by GENOUT.PUB based on the device selected. OUT_COLS is set to 0 if no device is OPENed (for example, when End is selected).
BBJPRINTER\$	Contains the java name of the BBj printer the form will print to
BBJPATH\$	Contains the name of the path that the PDF file will be printed to.
FORM_TYPE\$	The ID of the Form Type from the OSFRM file
COPIES	Number of copies to be printed (returned to calling program)

Variable	Description
PRINT_DEVICES\$	The device name of the printer or device for the output
PRINT_CHANNEL	The channel on which the printer will be opened
LINES	Unused
COPY_NAME\$[ALL]	The copy descriptions for the form (returned to calling program)
ALIGNMENT	Set to YES if an alignment should be printed; or NO if no alignment is necessary (returned to calling program)

GENOUT.PUB controls output selection, including printer selection, the file name for file output, and screen output. Terminal and printer initialization are performed if necessary. GENSCROL is ADDRed if screen output is selected.

The value of OUTCOLS controls the display of the standard or compressed selection message in the following manner:

- If a BBj printer is selected, the value will always be set to 132.
- If the terminal is capable and is defined correctly in the CONFIG.BBX and termcap files, it will produce compressed output.
- If the compressed-to-screen option for the workstation has been set to Inactive, the user will not be prompted for standard or compressed output. The system will use standard output.
- If the terminal is incapable of producing compressed output but compressed output is selected, processing will continue as if standard output had been selected. Reports that are sent to a file will have OUTCOLS set to 132, and the message will not be displayed. To disable the standard/compressed message on 80-column forms set OUTCOLS=-80, it will be returned as an absolute value to the workspace program.

GENOUT.PUB also determines whether GUI mode is currently active. If so, GENGOBBJ.PUB is run from GENOTBBJ.PUB. GENGOBBJ.PUB is the graphical version of GENOTBBJ.PUB, and allows for selection of device output, except within a GUI window.

GENOTHER.PUB

GENOTHER.PUB is a public program for processing the Other Commands key. This program is intended to be used only by GENINPUT.PUB.

Variable	Description
FILE[ALL]	Standard file array
MENUTEM	In GUI mode, the control ID of the menu selected

Exit Points	Explanation
EXIT	GENOTHER.PUB never returns an error.

GENOTHER.PUB displays a character menu and processes the menu(s) in OSMN. This file is maintained by specifying OS for the application code in the Menus function of Resource Manager. After a menu selection is performed, GENOTHER.PUB restores the state of the calling program and EXITS.

In GUI mode, GENOTHER.PUB will only process the selection, because the menu is always built into the current form.

GENOUT.PUB

GENOUT.PUB is a public program for selecting the output destination. This program is generally called from print programs.

```
XXXX ENTER DEVICE$,CHANNEL,OUT_COLS,FORM_TYPE$,COPIES,PRINT_DEVICES,
PRINT_CHANNEL,LINES,COPY_NAME$[ALL],ALIGNMENT
```

Variable	Description
DEVICE\$	Devices that can be selected. One or more of the following options are valid: E = all P = printer F = file S = Screen R = print preview W = Lotus worksheet output DEVICE\$ is set to the alias of the OPENed device upon exit.
CHANNEL	Set to the channel number of the device. If the channel comes in as 123, a WKS file type is added to the list of devices for Lotus output.
OUT_COLS	Number of output columns, usually 80 or 132. OUT_COLS can be changed by GENOUT.PUB based on the device selected. OUT_COLS is set to 0 if no device is OPENed (for example, when End is selected).
FORM_TYPE\$	The ID of the Form Type from the OSFRM file
COPIES	Number of copies to be printed (returned to calling program)
PRINT_DEVICES	The device name of the printer or device for the output
PRINT_CHANNEL	The channel on which the printer will be opened
LINES	Unused

Variable	Description
COPY_NAME\$[ALL]	The copy descriptions for the form (returned to calling program)
ALIGNMENT	Set to YES if an alignment should be printed; or NO if no alignment is necessary (returned to calling program)

GENOUT.PUB controls output selection, including printer selection, the file name for file output, and screen output. Terminal and printer initialization are performed if necessary. GENSCROL is ADDRed if screen output is selected.

The value of OUTCOLS controls the display of the standard or compressed selection message in the following manner:

- If the printer selected has been set up as a wide-carriage printer in the CONFIG.BBX file, the standard/compress message is always displayed.
- If the printer is defined as an 80-column printer, reports coded as 132 will always be compressed and no message will be displayed. The message is always displayed for reports defined as 80 and for reports sent to the screen.
- If the terminal is capable and is defined correctly in the CONFIG.BBX and termcap files, it will produce compressed output.
- If the compressed-to-screen option for the workstation has been set to Inactive, the user will not be prompted for standard or compressed output. The system will use standard output.
- If the terminal is incapable of producing compressed output but compressed output is selected, processing will continue as if standard output had been selected. Reports that are sent to a file will have OUTCOLS set to 132, and the message will not be displayed. To disable the standard/compressed message on 80-column forms set OUTCOLS=-80, it will be returned as an absolute value to the workspace program.

If a program needs to have multiple outputs, there are some restrictions. For example, if an audit log and an error report need to be produced in the same program, GENOUT.PUB can be called twice. However, if the first call allowed Printer or File to be selected and File was selected, the second selection would not allow file output even though DEVICE\$ had been set to "PF". Had the first selection been Printer, that printer's name would be removed from the available printer list, and file output would still be available. If only one printer were defined in the CONFIG.BBX file, printer output would not be displayed.

GENOUT.PUB also determines whether GUI mode is currently active. If so, GENGUOUT.PUB is run from GENOUT.PUB. GENGUOUT.PUB is the graphical version of GENOUT.PUB, and allows for selection of device output, except within a GUI window.

GENPC

GENPC is a general routine that CALLs GENPC.PUB to return the accounting period associated with a particular date, or to return the beginning and ending dates for an accounting period. GENPC has three entry points: GENPC for loading the table, GENPC1 for converting a month to a period, and GENPC2 for converting a period to beginning and ending dates. The routine resides at lines 9400–9499.

Below is the GENPC description:

Variable	Description
X\$	Company ID
O9\$	Set to string containing the conversion table data

Exit Points	Explanation
RETURN	GENPC normally returns.
GENERERROR	GENERERROR is invoked to handle errors.

GENPC need be called only once to initialize the O9\$ variable. The variable O9\$ is used to store a date conversion table. The data is retrieved from the “CNVTxxx” or “CNVT ” tables in the Resource Manager Tables (RMTB) file. If found, the company-specific table is used before the generic table.

Below is the GENPC1 description:

Variable	Description
X\$	Date mask stored in the global string "DATEMASK"
O9\$	Conversion table data
X	Set to the fiscal period for the date in X\$

Exit Points	Explanation
RETURN	GENPC1 normally returns.
GENERERROR	GENERERROR is invoked to handle errors.

The call to GENPC is automatic if needed by the GENPC1 call.

Here is the GENPC2 description:

Variable	Description
X\$	Period for which dates are computed
D\$	Date mask stored in the global string "DATEMASK"
O9\$	Conversion table data
BEGDATE	Set to the beginning date of the period
ENDDATE	Set to the ending date of the period

Exit Points	Explanation
RETURN	GENPC2 normally returns.
GENERERROR	GENERERROR is invoked to handle errors.

The call to GENPC is automatic if needed by the GENPC2 call. If you pass this routine a fiscal period, it attempts to return the beginning and ending dates. At this time it is used only in GL statement printing. You are encouraged to use it, however, if you need the functionality that it provides. Assign variables and GOSUB GENPC2:

```
3010 LET X=PD,D$=STBL("DATE")
3020 GOSUB GENPC2
3030 LET F6=BEGDATE,F7=ENDDATE
```

The year portion of the returned dates is taken from the current GL year. If no conversion table exists, the period beginning date is the first day of the current system date month; the period ending date is the current system date.

GENPC.PUB

GENPC.PUB is a public program that converts dates to periods, and vice versa.

```
9440 CALL "GENPC.PUB", FILE[ALL], SELECTION, DATE$, O9$, BEGDATE, ENDDATE, PERIOD, PPY
```

Variable	Description
FILE[ALL]	Standard file array
SELECTION	Determines the task GENPC.PUB is to perform: 0 = Return the period conversion table for the current company in O9\$ 1 = Return the period corresponding to the date provided 2 = Return the starting and ending dates for the period provided
DATE\$	The date for which the period is determined (SELECTION = 1)
O9\$	The period conversion table for this company
BEGDATE	GENOPEN.PUB and the CALLing program share the current command, in case the user aborts processing
ENDDATE	The ending date of the period provided (if SELECTION = 2)
PERIOD	The period to be converted (if SELECTION = 1 or SELECTION = 2), or the period corresponding to the date supplied
PPY	Number of periods in the company's fiscal year

Exit Points	Explanation
EXIT	GENPC.PUB normally exits with no error.
EXIT ERR	An error is returned if an error occurred.

The GENPC.PUB program automatically reads the RMTB file to find the period conversion table if O9\$ is blank. You do not need to open any table files to use GENPC.

GENPHONE

GENPHONE is a general routine that contains two multiline functions called FNPHONEMASK\$ and FNGUIPHONEMASK\$. These functions take a standard phone number and mask it using the specified mask. FNGUIPHONEMASK\$ also sets a mask for an INPUTE control. GENPHONE occupies lines 10100–10299 and should be merged into all programs that display or print phone numbers.

Variable	Description
XX	Any valid phone number
XX1\$	The phone number mask to be used
ID (FNGUIPHONEMASK\$ only)	A GUI control ID to which the masked data belongs

To use GENPHONE, substitute FNPHONEMASK\$(XX,XX1\$) for printing the phone number where it would print before.

```
7720 PRINT @(22,10),FNPHONEMASK$(PHONE$,MASK$),
```

instead of

```
7720 PRINT @(22,10),PHONE$,
```

You must make sure that you have the proper mask for the phone number (from the Country Code file) in MASK\$.

In GUI mode, the above code would look like this sample:

```
17720 PRINT (OSGUI.CHAN)'TITLE'(4010,FNGUIPHONEMASK$(PHONE$,MASK$,4010)
```

This would set the mask MASK\$ as the current mask for control ID 4010, and also display the data to the control.

GENPRBBJ.PUB

GENPRBBJ.PUB is a public program for displaying the available java printer selections that are configured to either the client workstation, or the printers that BBjServices can see on the server where BBjServices is running.

GENPRBBJ.PUB is generally called by GENOTBBJ.PUB. This program has two main areas: getting all printers available to BBj thru Java, and the actual selection process.

```
XXXX ENTER P_DEVICE$, P_CHANNEL, PREVIEW
```

Variable	Description
P_DEVICE\$	Set to the printer selected
P_CHANNEL	Not used
PREVIEW	Not used

Exit Points	Explanation
EXIT	GENPRNT.PUB does not return an error code.

Java functions are used to find out which printers are available to BBj. All printers that are set up on the client or server machine is displayed. This can be controlled thru Resource Manager, User Preferences, with the “BBj Forms Printers” option.

The printer selection uses GENINPUT.PUB. After a valid printer choice is made, the java name of the printer is returned thru the P_DEVICE\$ variable.

GENPRNT.PUB

GENPRNT.PUB is a public program for displaying the available printer selections as configured in the user's CONFIG.BBX file, except device PF, which is used to print to a file. GENPRNT.PUB is generally called by GENOUT.PUB. This program has two main areas: searching the CONFIG.BBX file for printers and the actual selection process.

```
XXXX ENTER P_DEVICE$, P_CHANNEL, PREVIEW
```

Variable	Description
P_DEVICE\$	Set to the printer selected
P_CHANNEL	Set to the channel number of the selected printer
PREVIEW	Set to 1 if Print Preview is selected, or 0 if not

Exit Points	Explanation
EXIT	GENPRNT.PUB does not return an error code.

The TSK() function is used to find out which devices are delivered in the CONFIG.BBX file. If the first character is **P** or **L**, the device name (except **PF**) is stored in the string array PRINTER\$(1:1:3) and appears in a window.

If this workstation has a default printer, the printer choice is set to this default. If there is only one valid printer, the user is not asked to choose it. The printer selection uses GENINPUT.PUB. After a valid printer choice is made, the printer channel is OPENed and control returns to the CALLing program.

GENPSCR.PUB

GENPSCR.PUB is a public program for printing the screen to a printer or file in text mode only. It is intended for the general routine GENERROR. No parameters are passed to GENPSCR.PUB.

Exit Points	Explanation
EXIT	GENPSCR.PUB never returns an error.

GENREPL.PUB

GENREPL.PUB is a protected public program used to replace user-defined text found in the current prefix. The CALL/ENTER list contains all the variables from the OSINFO.xxx file.

```
0100 ENTER INFODEV$, INFOPROG$, INFODATA1$, INFODATA2$, INFODATA3$, INFORW$,  
0100:INFOSAMP$, INFOUTIL$
```

Variable	Description
INFODEV\$	Installation/Backup device path
INFOPROG\$	Program directory
INFODATA1\$	Data directory 1
INFODATA2\$	Data directory 2
INFODATA3\$	Data directory 3
INFORW\$	Report Writer data directory
INFOSAMP\$	Sample data directory
INFOUTIL\$	Utilities directory
INFOSYS\$	System files directory

GENREPL.PUB looks for and replaces two patterns:

Pattern	Replaced with
%TERM%	Current Terminal ID (Txxx)
%USER%	STBL("USERDIRS")

The global variable STBL("USERDIRS") is built in RMSETUP. It uses values entered on the command line at the operating system prompt. Command line parameter **-d** is used to tell RMSETUP to use user-defined directories. This feature has not been fully integrated into the final product yet.

GENSAVE.PUB

GENSAVE.PUB is a public program used to save a workstation's status before releasing to the operating system. This program should be called just before executing a RELEASE verb with a parameter that will reinvoke OSAS.

```
0050 ENTER PROGRAM$,MENUSTATE$
```

Variable	Description
PROGRAM\$	Program name to return to when reentering OSAS
MENUSTATE\$	Current menu state before exiting (STBL("MENUSTATE"))

GENSAVE.PUB saves the following to the file RESTART.{workstation ID}:

Variable	Description
PROGRAM\$	Current program name
STBL("ACCESSCODE")	Current access code
STBL("JULDATE")	Current Julian date
STBL("DATE")	Current workstation date
MENUSTATE\$	Current menu state

RMSETUP looks for the existence of this file during startup and restores the data if the file is found.

If PROGRAM\$ is set to null, the workstation returns to the main menu after reentering OSAS. Otherwise, the workstation returns to the program from where it was released.

If PROGRAM\$ contains a value other than null, the global variable STBL("RESTART") must be cleared before running O1\$ (RMMENU) or the workstation will be returned to the program from where it was released. Place the following line somewhere before the line RUN O1\$ in the QUIT section of a program using GENSAVE.PUB:

```
9998 LET DUMMY$=STBL(" !CLEAR", "RESTART")
```

GENSCROL.PUB

GENSCROL.PUB is a public program for dealing with terminal information, setup, and movement for output to the screen. It creates the windows used in the report programs for terminal output of a report. It also monitors the keyboard, once a page of the report has been printed, and handles user input. It can manage five pages in memory at one time on a FIFO basis. If the workstation default for screen pages is increased, the user can keep more than five pages in memory. Increasing this number too much may cause unexpected errors if insufficient memory is available.

```
XXXX ENTER CHAN,WINDROW,EFLAG
```

Variable	Description
CHAN	Channel number for the terminal
WINDROW	First detail line of report (for positioning the highlighted bar)
EFLAG	End-of-Page or Report flag: 1 = end of page 2 = end of report

Exit Points	Explanation
EXIT	GENSCROL.PUB normally exits with no error.
EXIT 127	An Escape is returned if the user ESCAPEs (Ctrl+Break on PCs, or F7-Exit).

When the user presses **Enter**, the program creates a new window and returns to the CALLing program. Here is a list of acceptable keystrokes and the meaning of each:

Key	Movement
<Home>	First available page
<End>	Last available page
<PgUp>	Page backward in memory

Key	Movement
<PgDn>	Page forward in memory
<Up>	Highlight row up
<Down>	Highlight row down
<Right>	Move screen right one column
<Left>	Move screen left one column
<Tab>	Move screen page right/left
<Enter>	Print next NEW page (if last available page is not displayed, works like <PgDn>)

GENSELIN.PUB

GENSELIN.PUB is a public program for output selection prompts. This program is called by GENOUT.PUB.

```
XXXX ENTER PROMPT$,VAR$
```

Variable	Description
PROMPT\$	Text to print as the prompt. The prompt should be a series of choices with the first character of each choice enclosed by parentheses (for example, "(P)rinter (F)ile").
VAR\$	Set to the letter selected from the prompt string

Exit Points	Explanation
EXIT	GENSELIN.PUB does not return an error.

GENSELIN.PUB searches the PROMPT\$ string for parentheses. It keeps track of the position and value of the letters between the parentheses. To select an option, position the cursor at the letter of your choice. To move the cursor, use the arrow keys (which rotate between the choices) or press the first letter of your choice (the cursor moves to that letter) and then press **Enter**. The program exits with the selected value in VAR\$.

GENSORT.PUB

GENSORT.PUB is a public program for allocating two types of dynamic Mkeyed sort files. The first type is a sort file with a single key. The second type is a temporary Mkeyed file that duplicates the file to be sorted but with a different set of alternate keys available.

Variable	Description
CHAN	Channel number of the file to be sorted
SRTPRM[ALL]	Used to allocate key segments (described below)
STATUS	Set to -1 if the sort file allocation failed
FILENAME\$	Set to the file name of new sort file

Exit Points	Explanation
EXIT	GENSORT.PUB does not return any errors.

Here is an example of the data allocation and call statements:

```
xxxx DIM L[1:16,1:5]
xxxx LET L[X,Y]=VALUE, etc....
xxxx CALL "GENSORT.PUB",FILE[F0],L[ALL],X,F$
xxxx LET F0=7; GOSUB GENOPEN
```

To allocate a single-keyed sort file, assign the length of the key that you want the file sorted by to the first element of the L array—for example, L[1,1]=12. Call GENSORT and test X upon return. If the file was created, assign a file number to F0 and open it, using GENOPEN.

The file name will be "SCR"+FID(0), allowing for unique file names for all users. The file is created in the first data directory in the prefix, usually /osas/data. If the workstation default sort directory is used, the file will be placed in that directory first. The open and removal of this sort file must be done by the calling program.

The write statement for this type of sort file should look like this:

```
xxxx WRITE (FILE[F0],KEY=KEY VARIABLE,ERR=GENERROR)
```

No IOLIST is used for this file type. Only keys are stored in this file, which are made of concatenated pieces from the data. Some part of the key must point back to the original file in a unique fashion

If the initial read from the master file is made on one of the alternate keys, that key must still be the first part of the sort key to maintain that sort order. For example, to produce a report that lists employees by social security number and date of employment, the key length for the sort order would be 24 (11+7+6). The original file could be read by the alternate key for the social security number, but it still must be the first part of the sort key. The date of employment is next at a length of 7 for the Julian date, followed by the employee ID, which is the pointer back to the original file.

The second type of sort file is multi-keyed. Refer to the BB^x manual for a complete description of Mkeyed files and defining keys. To allocate a multi-keyed scratch file, assign the **L** array as follows (x is a number between 1 and 16):

L[x,1] is the key number you are defining and must be between 1 and 16. If the key number is the same for all 16 elements of the array, a single key with 16 concatenated segments is created. No key may exceed 64 bytes in length. The first or primary key must be unique.

L[x,2], the second element, must equal the number of the field in the original file. Its value should be between 1 and the number of the last field in the file. You can use a field number more than once.

L[x,3], the third element, must equal the starting position, or offset into the field, where the data will be found. Use a number between 1 and the length of the field. Keep in mind that the offset (**L[x,3]**) plus the length of the data piece (**L[x,4]**, see below) must be satisfied; otherwise, an error will occur.

L[x,4], the fourth element, must equal the length of the data piece starting at the offset (**L[x,3]**) and cannot exceed the length of the field.

L[x,5], the fifth element, has three settings. If it is set to 1, the key is arranged in descending order. If it is set to 2, the key is designated as unique. If it is set to 3, the key is arranged in descending order. If the key is made from concatenated segments, this element should be set on the first segment only; otherwise, an error will occur and no allocation will be performed.

This procedure creates a data file about the same size as the original file.

GENSORT2.PUB

GENSORT2.PUB is a public program that accepts a list of data, displays the data in a box, then prompts the user to select one of the specific options. This routine is used primarily for selecting alternate sort methods for several functions, including Sales Order and Purchase Order Order Inquiry, where the main HLE display can be sorted in several different ways.

Variable	Description
SORT\$[ALL]	Sorts to be displayed as prompts
SORT	Sort number that user selects
SORTNUM	Number of sorts defined within SORT\$[ALL]
OSGUI\$	GUI template (see GENGURES.PUB on page 3-57 for more details)

Exit Points	Explanation
EXIT	GENSORT.PUB does not return any errors.

GENSTDOC.PUB

GENSTDOC.PUB is a public program used to start up documents from applications. Users can set up documents for a variety of master files, including customers, vendors, and employees. To launch those documents, this routine is called so that documents are started in a consistent manner from within OSAS.

```
0060 ENTER FILE[ALL] , PATH$, FILE$, MENUPATH$, MSG$
```

Variable	Description
FILE[ALL]	Standard File array
PATH\$	Path where document exists
FILE\$	Document file to open
MENUPATH\$	Set only from menu programs to path in XXMN file when launching documents from menus
MSG\$	Error message returned from GENSTDOC.PUB

Exit Points	Explanation
EXIT	GENSTDOC.PUB does not return an error code.
EXITERR	GENSTDOC.PUB returns untrapped errors if an error occurs

This routine performs the following when opening up a document:

- Opens the OSFT file to determine the form type of the document, if one exists.
- Attempts to open the function used to start a document as defined by the OSFT file. If that can't be found, the OS default start command is used instead
- Attempts to open the document to be opened based on the PATH\$, FILE\$, and MENUPATH\$ variables. If the document is found, it's launched via the OSFT program or the OS default start program

GENTAX.PUB

GENTAX.PUB calculates sales tax based on the tax group, tax class, and price.

```
0025 ENTER FILE [ALL] , GROUP$ , CLASS$ , AMOUNT , TAX [ALL] , TYPE$ , ERATE , RMGC , RMTH , RMTD
```

Variable	Description
FILE[ALL]	Standard FILE[] array
GROUP\$	Tax group
CLASS\$	Tax class
AMOUNT	The sale or purchase price
TAX[ALL]	The tax array—DIM TAX[1:5,1:4] (values are returned by GENTAX.PUB)
TYPE\$	S for a sale or P for a purchase. Can add F for Freight, M for Misc., and A for Adjustment.
ERATE	Effective tax rate (returned by GENTAX.PUB)
RMGC	File index value for the RMGC file
RMTH	File index value for the RMTH file
RMTD	File index value for the RMTD file

Exit Points	Explanation
EXIT	GENTAX.PUB does not return an error.

The routine returns the TAX[ALL] array with the results of the calculation. The first DIMension of the array corresponds to the tax level (1–5) of each location in the group. The second DIMension contains four numeric values:

Elements	Value
1	Sales Tax Calculated
2	Refundable Amount Calculated
3	Taxable Sales/Purchases
4	Nontaxable Sales/Purchases

GENTAX.PUB also returns ERATE containing the effective tax percentage after all calculations.

GENTAX2.PUB

GENTAX2.PUB is a public program that updates the Tax Locations file with the amounts in the TAX[ALL] array. This routine is used in posting programs to update RMTHxxx and RMTDxxx.

```
0025 ENTER FILE[ALL],GROUP$,CLASS$,LOC$,ADJUST,TAX[ALL],TYPE$,RMGC,RMTD
```

Variable	Description
FILE[ALL]	Standard FILE[] array
GROUP\$	Tax group
CLASS\$	Tax class
LOC\$	Tax location for the adjustment
ADJUST	The adjustment amount
TAX[ALL]	The tax array set by GENTAX.PUB
TYPE\$	“S” for a sale or “P” for a purchase Can add “F” for Freight, “M” for Misc., and “A” for Adjustment
RMGC	File index value for the RMGC file
RMTD	File index value for the RMTD file

Exit Points	Explanation
EXIT	GENTAX2.PUB does not return an error.

GENUSR.PUB

GENUSR.PUB is a public program for processing the F4 User Command menu. This program is intended to be used only by RMMENU.

Variable	Description
FILE[ALL]	Standard file array
MENUITEM	In GUI mode, the control ID of the menu selected

Exit Points	Explanation
EXIT	GENOTHER.PUB never returns an error.

GENUSR.PUB displays a character menu and processes the F4USER menu in OSMN. This file is maintained by specifying OS for the application code in the Menus function of Resource Manager. After a menu selection is performed, control is returned to RMMENU, which in turn executes the selected program.

In GUI mode, GENUSR.PUB will only process the selection, because the menu is always built into the current form.

GENUSRFM.PUB

GENUSRFM.PUB is a public program that allows the user to add or view user defined fields that are set up for a given master file. Use this program if you have set up a separate file that will hold user-defined fields for any file you have.

```
0055 ENTER FILE[ALL],USR_DEF$,OSGUI$,PROMPT_NUM
```

Variable	Description
FILE[ALL]	Standard file array.
USER_DEF\$	Template containing setup information for this function.
OSGUI\$	GUI Template (see GENGURES.PUB on page 3-57 for more information).
PROMPT_NUM	Number of screen prompts for key fields.

Exit Points	Explanation
EXIT	GENUSRFM.PUB does not return any errors.

This chart breaks down the USER_DEF\$ template:

Field	Description
MAINT:C(1)	Set to “Y” if the function is to allow for edits.
INQUIRY:C(1)	Not used.
FILNAM:C(4)	File name used with which field definitions are stored.
MASTER[x]:N(2)	FILE[x] array on which the master file for xxUD file is opened.
TEMPLATE[x]:C(8)	Name of the template that the master file for xxUD is using.
DOC:N(2)	FILE[x] array element on which the xxUD file is opened.
TITLE:C(70)	The title that appears on the view screen.
USRTEMPL:C(8)	The name of the template for the xxUF file to use.
LABEL[x]:C(20)	Label that appears for each prompt.

Field	Description
ID[x]:C(20)	Data that appears for each prompt.
IDNAME[x]:C(20)	Not used.
NAME[x]:C(20)	Name of the field within TEMPLATE[x] that holds the data for this key field.
TASK:C(1)	Returns “X” if the user presses F5 to abandon the function, “N” if the user presses D (for Done); “S” if the user presses Page Up , or “M” if the user presses F7 to exit the function.
WINDID[x]:C(9)	Name of the F2 window to use when inquiring on this key field.
SELVAL[x]:C(20)	SELVAL used with the F2 window to use when inquiring on this key field.
TMP[x]:C(50)	Holds an expression used to print the data that appears on the screen.
TMP2[x]:C(50)	Holds an expression used to read data from the file opened on MASTER[x].

Note: The number passed in PROMPT_NUM determines how many prompt fields appear on the screen, and is used to prepare the USER_DEF\$ template.

When setting up to use this program, you can call GENUSRFM.PUB to have the USER_DEF\$ template set up for you. The template USER_DEF\$ has its arrays dimensioned to the number you pass in PROMPT_NUM.

```

5905 SET_TEMPLATE:
5907 SETERR GENERROR
5910 ENTER USER_DEF$,PROMPT_NUM

```

GENVWDOC.PUB

GENVWDOC.PUB is a public program that allows the user to view the user defined documents that are set up for a given master file. Use this program if you have set up a separate file that will hold documents for any master file you have.

```
0055 ENTER FILE[ALL],USER_VWDOC$,OSGUI$,PROMPT_NUM
```

Variable	Description
FILE[ALL]	Standard file array.
USER_VWDOC\$	Template containing setup information for this function.
OSGUI\$	GUI Template (see GENGURES.PUB on page 3-57 for more information).
PROMPT_NUM	Number of screen prompts for key fields.

Exit Points	Explanation
EXIT	GENVWDOC.PUB does not return any errors.

This chart breaks down the USER_VWDOC\$ template:

Field	Description
MASTER[x]:N(2)	FILE[x] array on which the master file for xxUD file is opened.
TEMPLATE[x]:C(8)	Name of the template that the master file for xxUD is using.
POS_KEY[x]:N(2)	Position within a string that the part of the master key is in, within the key string.
DOC:N(2)	FILE[x] array element on which the xxUD file is opened.
TITLE:C(70)	The title that appears on the view screen.
DOCTEMPL:C(8)	The template for the base application's xxUD file.
LABEL[x]:C(20)	Label that appears for each prompt.
ID[x]:C(20)	Data that appears for each prompt.

Field	Description
IDNAME[x]:C(20)	Not used.
NAME[x]:C(20)	Name of the field within TEMPLATE[x] that holds the data for this key field.
TASK:C(1)	Returns “X” when the user presses F7 to exit the function.
WINDID[x]:C(9)	Name of the F2 window to use when inquiring on this key field.
SELVAL[x]:C(20)	SELVAL used with the F2 window to use when inquiring on this key field.
TMP[x]:C(50)	Holds an expression used to print the data that appears on the screen.
TMP2[x]:C(50)	Holds an expression used to read data from the file opened on MASTER[x].

Note: The number passed in PROMPT_NUM determines how many prompt fields appear on the screen, and is used to prepare the USER_VWDOC\$ template.

When setting up to use this program, you can call GENVWDOC.PUB to have the USER_VWDOC\$ template set up for you. The template USER_VWDOC\$ has its arrays dimensioned to the number you pass in PROMPT_NUM.

```
5905 SET_TEMPLATE:
5907 SETERR GENERROR
5910 ENTER USER_VWDOC$,PROMPT_NUM
```

Variable	Description
USER_VWDOC\$	Template containing setup information for this function.
PROMPT_NUM	Number of screen prompts for key fields.

GENWIND.PUB

GENWIND.PUB is a public program for displaying inquiry windows. This program is intended to be used only by GENINPUT.PUB.

```
0100 ENTER Z9$,WINDID$,RETVAL$,SELVAL$,COMPID$,FILE[ALL]
```

Variable	Description
Z9\$	Unused
WINDID\$	Window ID of the inquiry window record in the xxWIND file
RETVAL\$	Starting value for entries in the window—set to the selected value when GENWIND.PUB exits
SELVAL\$	Selection value for limiting entries that appear in the window
COMPID\$	Unused
FILE[ALL]	Standard file array used for the data file channel number

Exit Points	Explanation
EXIT	GENWIND.PUB normally exits with no error.
EXIT ERR	An error is returned if an error occurs.

GENWIND.PUB performs these functions:

- OPENS the window data file, xxWIND
- builds list of alternate windows available
- reads the window record, KEY=WINDID\$
- displays a 'WINDOW'
- displays data records as defined in the window record

- accepts a command character
- allows the following cursor movement and record selection:
 - go to beginning/end of file
 - page up/down
 - one line up/down
 - select starting key to display
 - select alternate window to display
 - exit window and select the current key
 - exit window with no selection
 - display a help window

GENWIND.PUB traps all possible errors and returns them to the CALLing program. The routine uses the SETESC verb to prohibit the user from pressing **Ctrl+Break** and exiting from the program.

This routine does some sophisticated selection of key values to determine which keys should be displayed in the inquiry window. The following variables are used in the selection process:

- INCMASK\$ is the inclusion mask.
- EXCMASK\$ is the exclusion mask.
- SELVAL\$ is the selection value.

The selection process has two phases: the Inclusion test and the Exclusion test. Each test compares the file key value with the selection value character by character. A key won't be displayed in the inquiry window unless it is accepted by both tests.

The Inclusion test compares the file key value with the selection value, using the inclusion mask. The test fails when the character in the inclusion mask is \$FF\$ and the selection value doesn't equal the file key value. This test accepts keys that match the selection value in specific positions.

The Exclusion test compares the file key value with the selection value, using the exclusion mask. The test fails when the character in the exclusion mask is \$FF\$ and the selection value equals the file key value. This test accepts all the keys that don't match the selection value in the specified positions.

Inclusion/Exclusion Example

The inquiry window display for phase records from the JOBSxxx file shows how the inclusion and exclusion masks are used. For phase inquiry, the window should display only the job/phase records for the job already entered (the first 6 characters should equal the job ID). The window should not display the job record (the record with 6 trailing spaces).

The inclusion mask chooses all job/phase records with the same job ID. The exclusion mask excludes the job record.

Key Length = 12

Include mask begin (INCBEG) = 1
 Include mask length (INCLEN) = 6
 Exclude mask begin (EXCBEG) = 7
 Exclude mask length (EXCLEN) = 6

Selection value (SELVAL\$) = "20 " or \$32302020202020202020\$

Include mask (INCMASK\$) = \$FFFFFFFFF000000000000\$

Exclude mask (EXCMASK\$) = \$000000000000FFFFFFFFF\$

A bit-by-bit comparison is performed as follows:

```
X$=AND (XOR (FKEY$, SELVAL$) , INCMASK$)
X1$=IOR (AND (NOT (XOR (FKEY$, SELVAL$) ) , EXCMASK$) , NOT (EXCMASK$) )
```

These resulting strings are then tested for pass/fail.

The SELVAL\$ variable is used as follows:

If Inclusion is set at (1,6), the GENWIND.PUB program displays in the window only records that match the first six characters of the SELVAL\$ string.

If Exclusion is set at (7,6), and SELVAL\$="XXXXXX^~~~~", the GENWIND.PUB program does not display records having spaces in character positions 7–12.

You can use both Inclusion and Exclusion for the same window.

For example, if you set Inclusion = (1,6), Exclusion = (7,6), and SELVAL\$=R5\$(1,6)+FILL(6) (same as "JOB-ID^^^^"), only phases for the current job ID appear.

NOTE: This test is performed for other windows where the key to the data file comprises more than one data item.

GENWKS.PUB

GENWKS.PUB is a public program for outputting data to Lotus .WKS-type files. This program should be ADDRed before using it.

This program is a cosmetically modified version of the BB^x program _out123, which is distributed by BASIS Incorporated. Refer to the BB^x language reference manual for details about the use of this program.

NOTE: This program creates a .WKS file compatible with Lotus 1-2-3, version 1a or below. There is a limit of 2048 rows. If you are using a newer version of Lotus 1-2-3 and you want to use the larger capacity that these versions allow, change the workstation defaults for 123 Cell # and 123 Row # in Resource Manager.

This section describes the most common data files defined in OSAS, such as application files, company data files, sample data files and data dictionary files.

Application-Specific Routines

This section details the application-specific routines and public programs. Here is a list of the routines and a brief description of each.

Routine	Description
AP1099.PUB	Print 1099-MISC form
APACDCFM.PUB	Online distribution code maintenance
APACTCFM.PUB	Online terms code maintenance
APACVFM.PUB	Online vendor maintenance
APAGE.PUB	Age open invoices
APAUDIT.PUB	GL Account Audit Report for AP
APBLDQTY.PUB	Rebuild item quantities – AP Material Requisitions
APCJLIB.PUB	Library of CJ AP functions
APCMNT.PUB	Vendor comment inquiry and maintenance
APFOVCH.PUB	Print BBj PDF Voucher form
APHCDRIL.PUB	AP checks drill down driver
APHCTRAN.PUB	Write new records to the APHCx file
APHIDRIL.PUB	AP detail history drill down driver
APHITRAN.PUB	Write new records to the APHix file
APRESTRT.PUB	Check for aborted post or post in process
APVEINFO.PUB	Vendor information
APVOID.PUB	Prepare to void an Accounts Payable check
APVWDOC.PUB	Vendor document inquiry
ARACCDFM.PUB	Online direct debit maintenance
ARACCFM.PUB	Online customer maintenance
ARACCMFM.PUB	Online credit card maintenance
ARACDCFM.PUB	Online distribution account maintenance
ARACPCFM.PUB	Online payment method maintenance
ARACRFM.PUB	Online sales rep maintenance

Routine	Description
ARACSCFM.PUB	Online GL codes maintenance
ARACSVFM.PUB	Online shipping methods maintenance
ARACTCFM.PUB	Online terms codes maintenance
ARACTRAN.PUB	Write new records to the BARCx file
ARAGE.PUB	Age open invoices
ARAUDIT.PUB	GL Account Audit Report for AR
ARBLDQTY.PUB	Rebuild item quantities – AR Transactions
ARCMNT.PUB	Customer comment inquiry and maintenance
ARCRLCK1.PUB	Online credit-limit check
ARCRTRAN.PUB	Write new records to the ARCRx file
ARCUINFO.PUB	Customer information
ARDTINFO.PUB	Detail history information
ARFOINV.PUB	Print BBj PDF Invoice form
ARFOSTM.PUB	Print BBj PDF Statement form
ARHIDRIL.PUB	AR detail history drill down driver
ARHITRAN.PUB	Write new records to the ARHix file
ARPRIOP.PUB	Load templates for invoice printing
ARRESTRT.PUB	Check for aborted post or post in process
ARTRAN.PUB	Generate/Validate/Reserve Transaction Number
ARVWDOC.PUB	Customer document inquiry
BKAUDIT.PUB	GL Account Audit Report for BK
BKHIDRIL.PUB	BK detail history drill down driver
BKVWDOC.PUB	Bills of Materials document inquiry
BRAUDIT.PUB	GL Account Audit Report for BR
BRTRAN.PUB	Write Bank Reconciliation transaction
BRTRDRIL.PUB	BR transaction history drill down driver
CJAUDIT.PUB	GL Account Audit Report for CJ
CJHIDRIL.PUB	CJC detail history drill down driver
CJHITRAN.PUB	Write Contractors' Job Cost history records
CJHIDRIL.PUB	CJC detail history drill down driver
CJOSCNVT.PUB	Convert 7.5 Job Cost files to 7.5 CJC
CJVWDOC.PUB	Jobs document inquiry

Routine	Description
DDFOVCH.PUB	Print BBj PDF Voucher form
FAAUDIT.PUB	GL Account Audit Report for FA
FACOMP.PUB	Compute period depreciation
FAVWDOC.PUB	Asset document inquiry
GLACGFM.PUB	Online GL account entry
GLAUDIT.PUB	GL Account Audit Report for GL
GLJRN.L.PUB	Write General Ledger journal transaction
GLYEARS.PUB	Prepare file of active GL financial years
INACCLFM.PUB	Online customer level maintenance
INACGLFM.PUB	Online General Ledger codes maintenance
INACIFM.PUB	Online item maintenance
INACLFM.PUB	Online item location maintenance
INACSCFM.PUB	Online sales category maintenance
INACUFM.PUB	Online unit of measure maintenance
INAUDIT.PUB	GL Account Audit Report for IN
INBLDQTY.PUB	Rebuild item quantities – IN Transactions
INCALCPR.PUB	Calculates item sale price
INCMNT.PUB	Item comment inquiry and maintenance
INHCDRIL.PUB	IN COGS adjustment history drill down driver
INHCTRAN.PUB	Write new records to the INHCx file
INHIDRIL.PUB	IN detail history drill down driver
INITEM.PUB	Enter item ID with lookups
INQTYDSP.PUB	Display inventory quantities and quantity breaks
INSERTLOT.PUB	Enter and process serial and lot numbers
INSERTLT2.PUB	Enter and process serial and lot numbers
INSERTROL.PUB	Roll serial number file
INUPD	Program interface to INUPD.PUB
INUPD.PUB	Update Inventory files
INUPDSET.PUB	Set up for INUPD
INVWDOC.PUB	Item document inquiry
JOACCFM.PUB	Online cost code maintenance
JOACJCFM.PUB	Online job/phase cost code maintenance

Routine	Description
JOACJFM.PUB	Online job/phase maintenance
JOACTFM.PUB	Online job/phase cost type maintenance
JOAUDIT.PUB	GL Account Audit Report for JO
JOHIDRIL.PUB	JO detail history drill down driver
JOHITRAN.PUB	Write new record to the JOHIX file
JOVWDOC.PUB	Job/Phase document inquiry
PAACEFM.PUB	Online employee maintenance
PAACTGFM.PUB	Online tax group maintenance
PAAUDIT.PUB	GL Account Audit Report for PA
PACJCCFM.PUB	PA/CJC class code maintenance
PACJLIB.PUB	Library of CJ PA functions
PACMNT.PUB	Employee comment inquiry and maintenance
PAHCDRIL.PUB	PA check history drill down driver
PAHDDRIL.PUB	PA check deduction history drill down driver
PAHEDRIL.PUB	PA check earning history drill down driver
PAHWDRIL.PUB	PA check withholding history drill down driver
PAVOID.PUB	Void a payroll check
PAVWDOC.PUB	Employee document inquiry
PAYEARS.PUB	Prepare file of active PA Payroll years
POAUDIT.PUB	GL Account Audit Report for PO
POBLDQTY.PUB	Rebuild item quantities – PO Transactions
POFOPUR.PUB	Print BBj PDF Purchase Order form
POGENREQ.PUB	Generate Purchase Order requisitions
POGENSOQ.PUB	Generate Purchase Order requisitions from Sales Order
PORHDRIL.PUB	PO receipt history drill down driver
PORHTRAN.PUB	Write new record to the PORHX file
RMACBFM.PUB	Online batch maintenance
RMACGCFM.PUB	Online tax group maintenance
RMACTCFM.PUB	Online tax class maintenance
RMAUDIT.PUB	GL Account Audit Report for RM
RMCHKLY.PUB	Check for last-year GL files and prompt for current- or last-year entry
RMGETVER.PUB	Find application version number

Routine	Description
RMMERGE.PUB	Merge two data files
RMMNUMRG.PUB	Merge an application XXMNU.MRG text file into an XXMN file
RMPRTXT.PUB	Parse a text file in .INI format
RMUPCHGF.PUB	Merge application change field information into OSAS on application install
RMUPDICT.PUB	Merge application data dictionaries into OSAS on application install
RMUPGEN.PUB	Merge application GENERAL dictionary into OSAS on application install
SOAUDIT.PUB	GL Account Audit Report for SO
SOBLDQTY.PUB	Rebuild item quantities – SO Transactions
SOCRLCK1.PUB	Online credit limit check
SOKHDRIL.PUB	SO kit history drill down driver
SOPRIOP.PUB	Load templates for invoice printing
SOTRAN.PUB	Generate/Validate/Reserve Transaction Number

Routines ending with the extension .PUB are public programs. Access them with a CALL statement to the routine name in quotation marks:

```
xxxx CALL "ARACCFM.PUB", . . .
```

Routines with no extension are application-specific. You must merge them into a program and then access them with a GOSUB to the routine name (a label):

```
xxxx GOSUB INUPD
```

The application library (xxLIB) files on each application diskette contain the application-specific routines. You can use these files with the BASIS library manager to merge or update one or more application-specific routines in one or more programs.

AP1099.PUB

AP1099.PUB is a public program that prints out an AP 1099-MISC form.

```
0050 ENTER F,F1099_DATA$,LASER
```

Variable	Description
F	Channel on which the printer or file is open
F1099_DATA\$	String that stores information needed to print 1099-MISC
LASER	0 = Regular Forms 1 = Laser Forms

Exit Points	Explanation
EXIT	AP1099.PUB does not return any errors.

Once program flow enters AP1099.PUB, F1099_DATA\$ gets set into F1099\$, which is a template that takes this form:

Field	Description
VOID:C(1)	Set to 'X' to print a voided 1099-MISC
CORRECTED:C(1)	Set to 'X' to print corrected 1099-MISC (not used in OSAS)
PAYER[4]:C(30)	Company Name and Address
PFIN:C(15)	Company's Federal Identification Number
RECIPIENT[3]:C(30)	Recipient Name and Address
RIN:C(16)	Recipient Identification Number
BOX[1:12]:C(13)	Information for each box on 1099-MISC
ACCT_NUMBER:C(25)	Account Number (optional)
TIN_NOT:C(1)	Second TIN Notice
PARACHUTE:C(3)	Golden Parachute Payments (Not used in OSAS)
PHONE:C(12)	Payer's Phone Number

APACDCFM.PUB

APACDCFM.PUB is a public program that handles **F6 Maintenance** for distribution codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Distribution code entered by the user before pressing F6
O2\$	Accounts Payable or Purchase Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	APACDVFM.PUB does not return any errors.

APACTCFM.PUB

APACTCFM.PUB is a public program that handles **F6 Maintenance** for terms codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Terms code entered by the user before pressing F6
O2\$	Accounts Payable or Purchase Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	APACTCFM.PUB does not return any errors.

APACVFM.PUB

APACVFM.PUB is a public program that handles **F6 Maintenance** for vendors. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],VENDOR$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
VENDOR\$	Vendor ID entered by the user before pressing F6
O2\$	Accounts Payable or Purchase Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	APACVFM.PUB does not return any errors.

APAGE.PUB

APAGE.PUB is a public program that ages open invoices in the APINxxx file.

```
0100 ENTER
VEND$, INV$, AGEDATE, CUTOFF, BUCKETS [ALL] , DAYS [ALL] , APINFNUM, FILE [ALL] ,
TYPE, O2$, UNPDFIN, XPOS, YPOS, COUNT, HELD
```

Variable	Description
VEND\$	Vendor ID to age
INV\$	Specific invoice to age; set to "" to age all invoices for the vendor
AGEDATE	Aging date
CUTOFF	Cutoff date—invoices after the Julian date will not be aged; set to 0 to age all invoices
BUCKETS[ALL]	Returns aged balances in BUCKETS[1:6]
DAYS[ALL]	DIM DAYS[4] and set to aging days before calling (for example, DAYS[0]=0, DAYS[1]=30, DAYS[2]=60, and so on)
APINFNUM	FILE[] array index for the APINxxx file
FILE[ALL]	Standard file array used for the data file channel number
TYPE	Set to 1 for invoice date or 2 for due date
O2\$	Accounts Payable or Purchase Order options string
UNPDFIN	Unused
XPOS	Column position for speedo display; set to zero to suppress
YPOS	Row position for speedo display; set to zero to suppress
COUNT	Returns count of invoices processed
HELD	Set to 1 to ignore held invoices or 0 to include them

Exit Points	Explanation
EXIT	APAGE.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

APAUDIT.PUB

APAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Accounts Payable module.

FILE_CHECK:

This entry point returns the total number of files in Accounts Payable that has GL Accounts in them.

```
1510 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	APAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES, DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Accounts Payable.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of APAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: APDCx, APMDx, APPYx, APRLx, APTDx, and APVEx.

The tables that contain GL Accounts are: APGL, DF, DM, and MRGL.

APBLDQTY.PUB

APBLDQTY.PUB is a public program that creates a scratch file from AP Material Requisitions for use by the Inventory **Rebuild Item Quantities** function.

```
0060 ENTER FILE[ALL] , PASS$ , INQT$ , INSN$ , INVE$ , INLD$ , INLS$ , MSG$ , TASK , OPTION$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
PASS\$	Template array used for the data file channel number
INQT\$	INQTxxx file template (Quantity Totals file)
INSN\$	INSNxxx file template (Serial Numbers file)
INVE\$	INVExxx file template (Item Master file)
INLD\$	INLDxxx file template (Location Detail file)
INLS\$	INLSxxx file template (Transaction Lot/Serial Number file)
MSG\$	Error messages returned to calling program
TASK	0 = Open Files and prepare scratch file; 1 = Open Files only
OPTION\$	Application specific options and interfaces string

Exit Points	Explanation
EXIT	APBLDQTY.PUB does not return any errors.

The following chart breaks down the PASS\$ template.

Field	Description
INVE:N(2*)	Channel number on which the INVEx file is open
INLD:N(2*)	Channel number on which the INLDx file is open
QTSCR:N(2*)	Channel on which the INQTxxx scratch file is open (Quantity Totals file).

Field	Description
SNSCR:N(2*)	Channel on which the INSNxxx scratch file is open (Serial Number file)
ERRSCR:N(2*)	Channel on which the SCRT3.zzz error scratch file is open

APCJLIB.PUB

APCJLIB.PUB is a public program that handles a variety of Contractors' Job Cost functions from Accounts Payable, Purchase Order, and Landed Cost, via multiple entry points.

WIP_RANGE:

This entry point returns will read the JOBGL table from the CJTB file and return the WIP account range.

```
1005 WIP_RANGE:
1010 ENTER FILE[ALL] , WIP1$, WIP2$, CJWIPDF$, CJTB_INDEX
```

Variable	Description
FILE[ALL]	Standard file array
WIP1\$	WIP Account from value
WIP2\$	WIP Account thru value
CJWIPDF\$	Default WIP account
CJTB_INDEX	FILE[x] index that CJTB is opened on

Exit Points	Explanation
EXIT	APCJLIB.PUB exits with no error.

CHECK_GL_WIP

This entry point will check the GL Account you entered against the WIP range and let the user know if the data is valid.

```
1105 CHECK_GL_WIP:
1110 ENTER FILE[ALL] , OK$, WIP1$, WIP2$, GL_ACCT$, JOB_ID$, RET$, MSG$
```

Variable	Description
FILE[ALL]	Standard File Array
OK\$	Returns "Y" if GL Account is valid, blank if not valid

Variable	Description
WIP1\$	WIP account from
WIP2\$	WIP account thru
GL_ACCT\$	GL Account entered by the user to test
JOB_ID\$	Job ID entered by the user
RET\$	Retainage GL account from the APGL table
MSG\$	Returns an error message if GL account entered was not valid

CHECK_RET_INVOICE:

This entry point will check if the line item you entered is for a retainage invoice.

```
1205 CHECK_RET_INVOICE:
1210 ENTER FILE[ALL] , RET$, GL_ACCT$, VENDOR$, APIN_INDEX, JOB_ID$, OKAY$, RETAMT,
MSG$
```

Variable	Description
FILE[ALL]	Standard file array
RET\$	Retainage GL account number from the APGL table
GL_ACCT\$	GL account entered by the user
VENDOR\$	Vendor ID entered by the user
APIN_INDEX	FILE[x] index that APINx is opened on
JOB_ID\$	Job ID entered by the user
OKAY\$	“Y” = Retainage invoice entered “N” = No retainage invoice entered
RETAMT	Retainage amount for invoice
MSG\$	Message to display if invoice was not a retainage invoice

GET_GL_WIP:

This entry point will get the WIP GL account if Job/Phases are using specific WIP codes.

```

1305 GET_GL_WIP:
1310 ENTER FILE[ALL] , GL_ACCT$ , JOB_ID$ , B99$ [ALL] , CJWC_INDEX

```

Variable	Description
FILE[ALL]	Standard file array
GL_ACCT\$	GL WIP account from the CJWCx file
JOB_ID\$	Job/Phase ID entered by the user
B99\$[ALL]	WIP information for the Job/Phase ID entered
CJWC_INDEX	FILE[x] index the CJWCx file is opened on

JOB_FROM:

This entry point displays an entry prompt for Job ID from.

```

1405 JOB_FROM:
1410 ENTER FILE[ALL] , LISTYN$ , JOB_FROM$ , JOB_X0 , JOB_X1 , JOB_GUI , GUI , OSGUI$ , Y$

```

Variable	Description
FILE[ALL]	Standard file array
LISTYN\$	“Y” = Display message ‘Enter ‘*’ to input a list of jobs’
JOB_FROM\$	Job ID entered by the user
JOB_X0	X0 position of the Job From field used by GENINPUT
JOB_X1	X1 position of the Job Thru field used by GENINPUT
JOB_GUI	GUI ID of the Job From prompt used by GENINPUT
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template
Y\$	Command returned by GENINPUT.PUB

JOB_THRU:

This entry displays an entry prompt for Job ID thru.

```

1505 JOB_THRU:
1510 ENTER FILE[ALL] , JOB_FROM$ , JOB_X0 , JOB_X1 , JOB_GUI , JOB_THRU$ , GUI , OSGUI$ , Y$

```

Variable	Description
FILE[ALL]	Standard file array
JOB_FROM\$	Job From ID entered previously entered by the user
JOB_X0	X0 position of the Job From field used by GENINPUT
JOB_X1	X1 position of the Job Thru field used by GENINPUT
JOB_GUI	GUI ID of the Job From prompt used by GENINPUT
JOB_THRU\$	Job ID entered by the user
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template
Y\$	Command returned by GENINPUT.PUB

CLEAR_LIST:

This entry point will clear out the vendor or job list when preparing payments.

```
1605 CLEAR_LIST:
1610 ENTER FILE[ALL], LIST_TYPE$, GUI, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
LIST_TYPE\$	“V” = Clear vendor list “J” = Clear job list
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template

GET_LIST:

This entry point will paint and prompt for a list of either vendors or jobs.

```
1705 GET_LIST:
1710 ENTER FILE[ALL], FILE_INDEX, FROM_VALUE$, LIST_TYPE$, XXLIST$, XXLIST, GUI,
OSGUI$, Y$, X$
```

Variable	Description
FILE[ALL]	Standard file array
FILE_INDEX	FILE[x] element that APVEx or JOBSx is open on
FROM_VALUE\$	Last value entered in the list
LIST_TYPE\$	“V” = Vendor list “J” = Job list
XXLIST\$	Entire list of vendors or jobs entered
XXLIST	Position to start in list
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template
Y\$	Command returned by GENINPUT
X\$	Current vendor or job entered

GET_RET_ACCT:

This entry point will read the APGL table and return the retainage account.

```
2005 GET_RET_ACCT:
2010 ENTER FILE[ALL],TABLE_INDEX,GL_ACCT$
```

Variable	Description
FILE[ALL]	Standard file array
TABLE_INDEX	FILE[x] element that APTB, POTB, or LCTB is opened on
GL_ACCT\$	Retainage account from APGL table

UPDATE_JOB_TYPE:

This entry point will update the B5[ALL] cost types array with the correct amount

```
2105 UPDATE_JOB_TYPE:
2110 ENTER FILE[ALL],JOB_ID$,AMOUNT,B2[ALL],B5[ALL],SPREC
```

Variable	Description
FILE[ALL]	Standard file array
JOB_ID\$	Job/Phase/Cost Code entered by the user
AMOUNT	Extended amount of the line item
B2[ALL]	Array from JOBSx record containing overhead amount
B5[ALL]	Array from JOBSx record containing cost type amounts
SPREC	Precision for multiplication

RETAINAGE_INVOICE:

This entry point will process a retainage invoice for the Accounts Payable Post Transactions function.

```
2205 RETAINAGE_INVOICE:
2210 ENTER FILE[ALL],APTH_INDEX,APIN_INDEX,JOB_ID$,LINE_RETAIN[ALL],ORD_TYPE,
BATCH$,TRANS$,VENDOR$,FISCAL,INV_DATE,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
APTH_INDEX	FILE[x] index the APTHx file is opened on
APIN_INDEX	FILE[x] index the APINx file is opened on
JOB_ID\$	Job/Phase/Cost Code of the line item
LINE_RETAIN[ALL]	Retainage amounts from the line item
ORD_TYPE	"1" = Purchase "5" = Debit Memo
BATCH\$	Batch ID of the line item
TRANS\$	Transaction number of the line item
VENDOR\$	Vendor ID of the line item
FISCAL	Fiscal year this line is being posted to
INV_DATE	Invoice date
OSGUI\$	Standard GUI template

CHECK_ITEMS:

This entry point will check Purchase Order entry to let the user know if they need to apply retainage individually when dealing with retainage line items.

```
2405 CHECK_ITEMS:
2410 ENTER FILE[ALL] , POOR_INDEX, ORDER$, MSG$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
POOR_INDEX	FILE[x] index the POORx file is opened on
ORDER\$	Batch and order numbe of the transactions
MSG\$	Will return a message if entering a retainage line item
OSGUI\$	Standard GUI template

CHECK_PO_RETAINAGE

This entry point will check line items for retainage when applying invoices debit memos in Purchase Order Enter Order or Enter Returns.

```
2605 CHECK_PO_RETAINAGE:
2610 ENTER FILE[ALL] , POOR_INDEX, PORT_INDEX, INVC_NO$, INVOICE$, RET99[ALL] ,
GL_ACCT$, RET$, O2$, ORDER$, LOC2, WITH_RET$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
POOR_INDEX	FILE[x] index the POORx file is opened on
PORT_INDEX	FILE[x] index the PORTx file is opened on
INVC_NO\$	Original invoice number
INVOICE\$	Current invoice number
RET99[ALL]	Retainage amount
GL_ACCT\$	GL Account entered by the user
RET\$	Retainage GL account from the APGL table
O2\$	Applications option for PO or LC

Variable	Description
ORDER\$	Batch and order numbe of transaction
LOC2	Line item we are applying invoices to
WITH_RET\$	“Y” if we have retainage
OSGUI\$	Standard GUI template

PO_RETAINAGE_INVOICE:

This entry point will process a retainage invoice for the Purchase Order Post Transactions function.

```
2805 PO_RETAINAGE_INVOICE:
2810 ENTER FILE[ALL] , APIN_INDEX, XX99 [ALL] , XX99$ [ALL] , TRET, BSUB, VENDOR$, E2,
INVDATA, DUEDATE, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
APIN_INDEX	FILE[x] index the APINx file is opened on
XX99[ALL]	Retainage amounts
XX99\$[ALL]	Job/Phase/Cost Code
TRET	Total Retainage
BSUB	Invoice Subtotal
VENDOR\$	Vendor ID of the invoice being processed
E2	Date of order. Set to -1 for debit memos
INVDATA	Invoice date
DUEDATE	Invoice due date
OSGUI\$	Standard GUI template

RETENTION:

This entry point will display and prompt the user for retainage amount and retainage percent for Accounts Payable, Purchase Order, and Landed Cost.

```
3105 RETENTION:
3110 ENTER FILE[ALL] , Y$, RETAIN_ID, PERCENT_ID, JOB_ID$, GL_ACCT$, RET$, EXT_COST,
XX99[ALL] , OLDRET, RET_X, RET_Y, PER_X, PER_Y, M2$, PERCENT, GUI, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
Y\$	Command returned by GENINPUT
RETAIN_ID	GUI ID of the retainage prompt for GENINPUT
PERCENT_ID	GUI ID of the percent prompt for GENINPUT
JOB_ID\$	Job ID entered by the user
GL_ACCT\$	GL account entered by the user
RET\$	Retainage GL account from the APGL table
EXT_COST	Extended cost amount entered by the user
XX99[ALL]	Retainage amount entered by the user
OLDRET	Original retainage amount entered by the user
RET_X	X1 position of the Retainage prompt for GENINPUT
RET_Y	X0 position of the Retainage prompt for GENINPUT
PER_X	X1 position of the Retainage Percent prompt for GENINPUT
PER_Y	X0 position of the Retainage Percent prompt for GENINPUT
M2\$	STLB("DOLL") mask
PERCENT	Vendor default retainage percent
GUI	1 if in GUI mode
OSGUI\$	Standard GUI template

GET_JOB

This entry point will calculate and return the retainage amount left for purchase orders and returns when finishing invoice or debit memo entry.

```
3505 GET_JOB:
3510 ENTER FILE[ALL] , PORI_INDEX, PORT_INDEX, POOR_INDEX, INVOICE$, ORDER$,
RET99$[ALL] , RET99 [ALL] , RET$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
PORI_INDEX	FILE[X] index the PORIx file is opened on
PORT_INDEX	FILE[X] index the PORTx file is opened on
POOR_INDEX	FILE[X] index the POORx file is opened on
INVOICE\$	Invoice or debit memo number we are processing
ORDER\$	Batch and order number of the current transaction
RET99\$[ALL]	Job/Phase/Cost Code returned
RET99[ALL]	Retainage amounts returned
RET\$	Retainage GL account from the APGL table
OSGUI\$	Standard GUI template

APCMNT.PUB

APCMNT.PUB is a public program that provides entry into GENCMNT.PUB for vendor comment inquiry and maintenance.

```
0050 ENTER FILE[ALL],VENDID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
VENDID\$	Vendor ID for which comments are displayed and edited
X0\$	A = Maintenance mode; B = Inquiry only
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	APCMNT.PUB does not return any errors.

APFOVCH.PUB

APFOVCH.PUB is a public program that will generate a BBj PDF voucher form from properly formatted data files. This program will take the data and produce an Accounts Payable/Banking voucher looking similiar to the OSAS laser form.

PRINT_FORM:

This entry point is used to print an invoice form after data is set up in temporary files.

```
0050 PRINT_FORM:
0060 ENTER VCHINFO$,APVFH$,APVFD$,FILE[ALL]
```

Variable	Description
VCHINFO\$	Template used for generic form information
APVFH\$	Template for voucher header temporary file
APVFD\$	Template for voucher detail temporary file
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	APFOVCH.PUB normally exits with no error.

GET_TEMPLATES:

This entry point is to retrieve the template format that will be set up and used for general form information, as well as the temporary files built for use for printing vouchers.

```
4010 GET_TEMPLATES;
4020 ENTER VCHINFO$,APVFH$,APVFD$
```

Variable	Description
VCHINFO\$	Template used for generic form information
APVFH\$	Template for voucher header temporary file
APVFD\$	Template for voucher detail temporary file

The following chart breaks down the VCHINFO\$ template.

Variable	Description
PRINTNAME:C(30*)	BBj Java name of printer to print to
FORMID:C(8*)	Form ID from scratch files to print
OUTPATH:C(65*)	Path and name of PDF file to generate if e-mailing or printing to file
LOGO:C(80*)	Path and name of logo image to print on form
COPYTEXT:C(15*)	Text to print if printing multiple copies of form
INDEXHEAD:N(2*)	FILE[x] element that header scratch file is opened on (ARIFHxxx)
INDEXDETL:N(2*)	FILE[x] element that detail scratch file is opened on (ARIFDxxx)
STUBNAME:C(30*)	Title of stub
COMPNAME:C(30*)	Current company's name
COMPADD1:C(30*)	Current company's Address Line 1
COMPADD2:C(30*)	Current company's Address Line 2
COMPADD3:C(30*)	Current company's Address Line 3
COMPADD4:C(30*)	Current company's Address Line 4
BANKADD1:C(30*)	Bank's Address Line 1
BANKADD2:C(30*)	Bank's Address Line 2
BANKADD3:C(30*)	Bank's Address Line 3
BANKADD4:C(30*)	Bank's Address Line 4
BANKADD5:C(30*)	Bank's Address Line 5
LOGOP:C(2*)	L=Left justified, other values = Centered
FORMPROG:C(12*)	"APFOVCH.PUB". Set in a variable so that the multi-copy form printing program knows what program to call to print invoices

See the Banking file layouts for APVFBHxxx and APVFDxxx for layouts of the temporary files and their templates.

APHCDRIL.PUB

APHCDRIL.PUB is a public program that handles the drill down ability from General Ledger to Accounts Payable Check History information. This program will build a scratch file containing matching historical entries between the GLDRx and APHCx files, then call APHCICON to display those records.

```
0060 ENTER FILE[ALL], OSGUI$, GLJRSEQ$, YEAR_EXT$, MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	APHCDRIL.PUB does not return any errors.

APHCTRAN.PUB

APHCTRAN.PUB is a public program that handles writing records to the Accounts Payable check history file. This interface is called from routines that need to write new records to APHCx.

(Normal entry point)

This entry point is used when new records are to be written to the APHCx file. If HC0\$ is set prior to calling APHCTRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If HC0\$ is set to “”, a new sequence number will be generated and returned thru HC0\$.

```
0099 ENTER F0,HC0$,HC1$,HC2$,HC1,HC3$,HC4$,HC5$,HC6$,HC2,HC7$,HC8$,HC9$,
HC10$,HC3,HC99$[ALL],HC99[ALL]
```

Variable	Description
F0	Channel APHCx file is opened on
HC0\$	If set prior to calling APHCTRAN.PUB, that sequence will always be used when writing a new record to APHCx. If set to “”, the sequence number generated will be returned to the calling program
HC1\$,..HC99[ALL]	APHCxxx IOList variables (see the Accounts Payable File Layouts)

Exit Points	Explanation
EXIT	APHCTRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_APHC_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the NXTSEQ record in APHCx.


```
3010 RESERVE_APHI_SEQ:  
3015 ENTER F0,SEQ$
```

Variable	Description
F0	Channel APHCx file is opened on
SEQ\$	Next available sequence number in APHCx

APHIDRIL.PUB

APHIDRIL.PUB is a public program that handles the drill down ability from General Ledger to Accounts Payable Detail History information. This program will build a scratch file containing matching historical entries between the GLDRx and APHix files, then call either APHINCON if the user is drilling on totals records, or APSDHCON if the user is drilling on line item or payment records.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	APHIDRIL.PUB does not return any errors.

APHITRAN.PUB

APHITRAN.PUB is a public program that handles writing records to the Accounts Payable detail history file. This interface is called from routines that need to write new records to APHIX.

(Normal entry point)

This entry point is used when new records are to be written to the APHIX file. If H5\$ is set prior to calling APHITRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If H5\$ is set to “”, a new sequence number will be generated and returned thru H5\$.

```
0099 ENTER F0,H5$,H0$,H1$,H2$,H0,H1,H2,H3,H3$,H4$,H4,H5,H6,H7,H8,H9[ALL],H6$,  
H7$,H99$[ALL],H99[ALL]
```

Variable	Description
F0	Channel APHIX file is opened on
H5\$	If set prior to calling APHITRAN.PUB, that sequence will always be used when writing a new record to APHIX. If set to “”, the sequence number generated will be returned to the calling program
H1\$,H2\$,..H99[ALL]	APHIXxx IOList variables (see the Accounts Payable File Layouts)

Exit Points	Explanation
EXIT	APHITRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_APHI_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the NXTSEQ record in APHIX.

```
3010 RESERVE_APHI_SEQ:  
3015 ENTER F0,SEQ$
```

Variable	Description
F0	Channel APHIX file is opened on
SEQ\$	Next available sequence number in APHIX

APRESTRT.PUB

APRESTRT.PUB is a public program that checks for aborted posts and posts in process on another terminal.

```
0100 ENTER F0,F1,BATCH$,TERM$,MSG$,PICKS$,PARAM,STATUS
```

Variable	Description
F0	Channel number of the APBTxxx file
F1	Channel number of the APTMTxxx file; set to 0 if not opened
BATCH\$	Batch ID to check
TERM\$	Terminal ID to check or FILL(4) to check all terminals
MSG\$	Returns a message string if the check fails
PICKS\$	Returns the pick screen options to restart the post
PARAM	Task to perform: 0 - check for aborted post on the terminal 1 - check for aborted post on all terminals 2 - check for batch in use on another terminal 3 - change print status flag for the batch 4 - create and lock temporary print lock file
STATUS	Returns a value based on the batch status: 0 - batch is available 1 - aborted post detected 2 - batch posting on another terminal 3 - aborted print job detected 4 - batch printing on another terminal 5 - batch ID not found 6 - unable to create or lock temporary post or print file -99 - error updating print status

Exit Points	Explanation
EXIT	APRESTRT.PUB does not return an error.

APVEINFO.PUB

APVEINFO.PUB is a public program that displays vendor information when accessed through **Shift+F2** in Accounts Payable or Purchase Order transactions.

```
0200 ENTER FILE[ALL],DEFAULT$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
DEFAULT\$	Default vendor ID with which to start
X0\$	Not used
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	APVEINFO.PUB does not return any errors.

APVOID.PUB

APVOID.PUB is a public program that sets up to void a check from AP. This program is called from within Bank Reconciliation **Transactions** if the check to be voided is from Accounts Payable.

```
0050 ENTER MSG$,CHECKNO$,AMOUNT,RETURN$,A$,DATE,OSGUI$
```

Variable	Description
MSG\$	Returns a message string if a trapped error occurs
CHECKNO\$	Check number to void; if blank, user will be prompted
AMOUNT	Returns the amount of the check
RETURN\$	Vendor ID associated with the check; if blank, user is prompted
A\$	Application ID (AP or PO)
DATE	Date of the check; if blank, the user will be prompted
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	APVOID.PUB does not return an error.

APVWDOC.PUB

APVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for vendor document inquiry.

```
ENTER FILE[ALL], VENDID$, X0$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
VENDID\$	Vendor ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	APVWDOC.PUB does not return any errors.

ARACCDFM.PUB

ARACCDFM.PUB is a public program that handles **F6 Maintenance** for customer direct debit accounts. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Direct debit account number entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACCDFM.PUB does not return any errors.

ARACCFM.PUB

ARACCFM.PUB is a public program that handles **F6 Maintenance** for customers. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CUSTOMER$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CUSTOMER\$	Customer ID entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACCFM.PUB does not return any errors.

ARACCMFM.PUB

ARACCMFM.PUB is a public program that handles **F6 Maintenance** for customer credit cards. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE [ALL] , CODE$, O2$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Credit card number entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACCMFM.PUB does not return any errors.

ARACDCFM.PUB

ARACDCFM.PUB is a public program that handles **F6 Maintenance** for distribution accounts. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Distribution code entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACDCFM.PUB does not return any errors.

ARACPCFM.PUB

ARACPCFM.PUB is a public program that handles **F6 Maintenance** for payment methods. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Payment method entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACPCFM.PUB does not return any errors.

ARACRFM.PUB

ARACRFM.PUB is a public program that handles **F6 Maintenance** for sales reps. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL], REP$, O2$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
REP\$	Sales Rep ID entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACRFM.PUB does not return any errors.

ARACSCFM.PUB

ARACSCFM.PUB is a public program that handles **F6 Maintenance** for GL codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE [ALL] , CODE$ , O2$ , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	GL code entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACSCFM.PUB does not return any errors.

ARACSVFM.PUB

ARACSVFM.PUB is a public program that handles **F6 Maintenance** for shipping methods. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE [ALL] , CODE$ , O2$ , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Shipping method entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACSVFM.PUB does not return any errors.

ARACTCFM.PUB

ARACTCFM.PUB is a public program that handles **F6 Maintenance** for terms codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Terms code entered by the user before pressing F6
O2\$	Accounts Receivable or Sales Order options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARACTCFM.PUB does not return any errors.

ARACTRAN.PUB

ARACTRAN.PUB is a public program that handles writing records to the Accounts Receivable ACH file. This interface is called from routines that need to write new records to BARC_x.

The NXTSEQ record from the BARC_x file is read to determine the next available sequence number for the ACH file, which is then used to write the new record to BARC_x. That sequence number is also increased by 1, and written back to the NXTSEQ record in BARC_x.

```
0090 ENTER F0,BARC$
```

Variable	Description
F0	Channel BARC _x file is open on
BARC\$	Template for BARC _x file (see the Banking File Layouts)

Exit Points	Explanation
EXIT	ARACTRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

ARAGE.PUB

ARAGE.PUB is a public program that ages open invoices in the ARINxxx file.

```
0100 ENTER CUST$, INV$, AGEDATE, CUTOFF, BUCKETS [ALL] , DAYS [ALL] , ARINFNUM, FILE [ALL] ,
0100:TYPE, O2$, UNPDFIN, XPOS, YPOS, COUNT, HELD, OSGUI$
```

Variable	Description
CUST\$	Customer ID to age
INV\$	Specific invoice to age; set to "" to age all invoices for the customer
AGEDATE	Aging date
CUTOFF	Cutoff date. Invoices after the Julian date are not aged; set to 0 to age all invoices
BUCKETS[ALL]	Returns aged balances in BUCKETS[1:6]
DAYS[ALL]	DIM DAYS[4] and set to aging days before calling (for example, DAYS[0]=0, DAYS[1]=30, DAYS[2]=60, and so on)
ARINFNUM	FILE[] array index for the ARINxxx file
FILE[ALL]	Standard file array used for the data file channel number
TYPE	Set to 1 for invoice date or 2 for due date
O2\$	Accounts Receivable or Sales Order options string
UNPDFIN	Unused
XPOS	Column position for speedo display; set to 0 to suppress
YPOS	Row position for speedo display; set to 0 to suppress
COUNT	Returns count of invoices processed
HELD	Set to 1 to ignore held invoices or 0 to include them
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARAGE.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

ARAUDIT.PUB

ARAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Accounts Receivable module.

FILE_CHECK:

This entry point returns the total number of files in Accounts Receivable that has GL Accounts in them.

```
1010 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	ARAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Accounts Receivable.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of ARAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: ARCRx, ARDCx, ARGLx, ARPYx, ARREx and ARTDx. If Contractors' Job Cost is installed, we will also check GL accounts in the following files: ARJAx, ARJBx, ARJDx, and ARJEx.

The table that contain GL Accounts is the ARGLx table.

ARBLDQTY.PUB

ARBLDQTY.PUB is a public program that creates a scratch file from the AR Transactions file for use by the Inventory **Rebuild Item Quantities** function.

```
0060 ENTER FILE[ALL],PASS$,INQT$,INSN$,INVE$,INLD$,INLS$,MSG$,TASK,OPTION$
```

Variable	Description
FILE[ALL]	Standard file array used for the file channel number
PASS\$	Template array used for the data file channel number
INQT\$	INQTxxx file template (Quantity Totals)
INSN\$	INSNxxx file template (Serial Numbers)
INVE\$	INVExxx file template (Item Master)
INLD\$	INLDxxx file template (Location Detail)
INLS\$	INLSxxx file template (Transaction Lot/Serial Number)
MSG\$	Error messages returned to calling program
TASK	0 = Open files and create sort file; 1 = Open files only
OPTION\$	Application-specific options and interfaces string

Exit Points	Explanation
EXIT	ARBLDQTY.PUB does not return any errors.

The following chart breaks down the PASS\$ template.

Variable	Description
INVE:N(2*)	Channel number on which the INVEx file is open
INLD:N(2*)	Channel number on which the INLDx file is open
QTSCR:N(2*)	Channel on which the INQTxxx scratch file is open (Quantity Totals file)
SNSCR:N(2*)	Channel on which the INSNxxx scratch file is open (Serial Number file)
ERRSCR:N(2*)	Channel on which the SCRT3.zzz error scratch file is open

ARCMNT.PUB

ARCMNT.PUB is a public program that provides entry into GENCMNT.PUB for customer comment inquiry and maintenance.

```
0050 ENTER FILE[ALL],VENDID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
VENDID\$	Customer ID for which comments are displayed and edited
X0\$	A = Maintenance mode B = Inquiry only
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARCMNT.PUB does not return any errors.

ARCRLCK1.PUB

ARCRLCK1.PUB is a public program that checks the customer balance against the credit limit. It includes current transactions and cash receipts in the balance.

```
0100 ENTER CUST$,CREDLIM,ARTHFNUM,ARCRFNUM,FILE[ALL],CURRTRAN$
```

Variable	Description
CUST\$	Customer ID
CREDLIM	Customer credit limit
ARTHFNUM	FILE[] array index for the ARTHxxx file
ARCRFNUM	FILE[] array index for the ARCRxxx file
FILE[ALL]	Standard file array used for the data file channel number
CURRTRAN\$	Current transaction number

Exit Points	Explanation
EXIT	ARCRLCK1.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

ARCRTRAN.PUB

ARCRTRAN.PUB is a public program that handles writing records to the Accounts Receivable cash receipt file. This interface is called from routines that need to write new records to ARCRx.

The NXCx record from the ARCTx file is read to determine the next available sequence number for the cash receipt file, which is then used to write the new record to the ARCRx file. That sequence number is also increased by 1, and written back to ARCTx.

```
0090 ENTER ARCRCH,ARCTCH,CR0$,CR1$,CR1[ALL],CR2$,CR3$,CR2[ALL],CR4$,CR5$,  
CR6$,CR99$[ALL],CR99[ALL]
```

Variable	Description
ARCRCH	Channel ARCRx file is open on
ARCTCH	Channel ARCTx file is open on
CR0\$,..CR99[ALL]	ARCRxxx IOList variables (see the Accounts Receivable File Layouts)

Exit Points	Explanation
EXIT	ARCRTRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

ARCUINFO.PUB

ARCUINFO.PUB is a public program that displays customer information when accessed through **Shift+F2** in Accounts Receivable and Sales Order transactions.

```
0200 ENTER FILE[ALL],DEFAULT$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
DEFAULT\$	Default customer ID with which to start
X0\$	Not used
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARCUINFO.PUB does not return any errors.

ARDTINFO.PUB

ARDTINFO.PUB is a public program that displays detail history information for customers when accessed through **Shift+F2** in Accounts Receivable and Sales Order transactions.

```
0200 ENTER FILE[ALL],DEFAULT$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
DEFAULT\$	Default customer ID with which to start
X0\$	Not used
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	ARDTINFO.PUB does not return any errors.

ARFOINV.PUB

ARFOINV.PUB is a public program that will generate a BBj PDF invoice form from properly formatted data files. This program will take the data and produce an invoice looking similiar to the OSAS laser form.

PRINT_FORM:

This entry point is used to print an invoice form after data is set up in temporary files.

```
0050 PRINT_FORM:
0060 ENTER INVCINFO$,ARIFH$,ARIFD$,FILE[ALL]
```

Variable	Description
INVCINFO\$	Template used for generic form information
ARIFH\$	Template for invoice header temporary file
ARIFD\$	Template for invoice detail temporary file
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	ARFOINV.PUB normally exits with no error.

GET_TEMPLATES:

This entry point is to retrieve the template format that will be set up and used for general form information, as well as the temporary files built for use for printing invoices.

```
4010 GET_TEMPLATES;
4020 ENTER INVCINFO$,ARIFH$,ARIFD$
```

Variable	Description
INVCINFO\$	Template used for generic form information
ARIFH\$	Template for invoice header temporary file
ARIFD\$	Template for invoice detail temporary file

The following chart breaks down the INVCINFO\$ template.

Variable	Description
PRINTNAME:C(30*)	BBj Java name of printer to print to
FORMID:C(8*)	Form ID from scratch files to print
OUTPATH:C(65*)	Path and name of PDF file to generate if e-mailing or printing to file
LOGO:C(80*)	Path and name of logo image to print on form
COPYTEXT:C(15*)	Text to print if printing multiple copies of form
INDEXHEAD:N(2*)	FILE[x] element that header scratch file is opened on (ARIFHxxx)
INDEXDETL:N(2*)	FILE[x] element that detail scratch file is opened on (ARIFDxxx)
COMPNAME:C(30*)	Current company's name
COMPADD1:C(30*)	Current company's Address Line 1
COMPADD2:C(30*)	Current company's Address Line 2
COMPADD3:C(30*)	Current company's Address Line 3
COMPADD4:C(30*)	Current company's Address Line 4
COMPADD5:C(30*)	Current company's Address Line 5
LOGOP:C(2*)	L=Left justified, other values = Centered
FORMPROG:C(12*)	"ARFOINV.PUB". Set in a variable so that the multi-copy form printing program knows what program to call to print invoices

See the Accounts Receivable file layouts for ARIFHxxx and ARIFDxxx for layouts of the temporary files and their templates.

ARFOSTM.PUB

ARFOSTM.PUB is a public program that will generate a BBj PDF statement form from properly formatted data files. This program will take the data and produce a statement looking similiar to the OSAS laser form.

PRINT_FORM:

This entry point is used to print an invoice form after data is set up in temporary files.

```
0050 PRINT_FORM:
0060 ENTER STMTINFO$, ARSFH$, ARSFD$, FILE[ALL]
```

Variable	Description
STMTINFO\$	Template used for generic form information
ARSFH\$	Template for statement header temporary file
ARSFD\$	Template for statement detail temporary file
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	ARFOSTM.PUB normally exits with no error.

GET_TEMPLATES:

This entry point is to retrieve the template format that will be set up and used for general form information, as well as the temporary files built for use for printing statements.

```
4010 GET_TEMPLATES;
4020 ENTER STMTINFO$, ARSFH$, ARSFD$
```

Variable	Description
STMTINFO\$	Template used for generic form information
ARSFH\$	Template for statement header temporary file
ARSFD\$	Template for statement detail temporary file

The following chart breaks down the STMTINFO\$ template.

Variable	Description
PRINTNAME:C(30*)	BBj Java name of printer to print to
FORMID:C(8*)	Form ID from scratch files to print
OUTPATH:C(65*)	Path and name of PDF file to generate if e-mailing or printing to file
LOGO:C(80*)	Path and name of logo image to print on form
COPYTEXT:C(15*)	Text to print if printing multiple copies of form
INDEXHEAD:N(2*)	FILE[x] element that header scratch file is opened on (ARIFHxxx)
INDEXDETL:N(2*)	FILE[x] element that detail scratch file is opened on (ARIFDxxx)
COMPNAME:C(30*)	Current company's name
COMPADD1:C(30*)	Current company's Address Line 1
COMPADD2:C(30*)	Current company's Address Line 2
COMPADD3:C(30*)	Current company's Address Line 3
COMPADD4:C(30*)	Current company's Address Line 4
COMPADD5:C(30*)	Current company's Address Line 5
AGE1HEAD:C(15*)	Column heading for first aging bucket
AGE2HEAD:C(15*)	Column heading for second aging bucket
AGE3HEAD:C(15*)	Column heading for third aging bucket
AGE4HEAD:C(15*)	Column heading for fourth aging bucket
LOGOP:C(2*)	L=Left justified, other values = Centered
FINCHG:N(1*)	Set to 1 if printing YTD finance charges on statement
FORMPROG:C(12*)	"ARFOSTM.PUB". Set in a variable so that the multi-copy form printing program knows what program to call to print invoices

See the Accounts Receivable file layouts for ARSFHxxx and ARSFDxxx for layouts of the temporary files and their templates.

ARHIDRIL.PUB

ARHIDRIL.PUB is a public program that handles the drill down ability from General Ledger to Accounts Receivable detail history information. This program will build a scratch file containing matching historical entries between the GLDRx and ARHix files, then call either ARHINCON if the user is drilling on totals records, or ARSDHCON if the user is drilling on line item, payment, or finance charge records.

```
0060 ENTER FILE[ALL], OSGUI$, GLJRSEQ$, YEAR_EXT$, MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	ARHIDRIL.PUB does not return any errors.

ARHITRAN.PUB

ARHITRAN.PUB is a public program that handles writing records to the Accounts Receivable detail history file. This interface is called from routines that need to write new records to ARHix.

(Normal entry point)

This entry point is used when new records are to be written to the ARHix file. If H0\$ is set prior to calling ARHITRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If H0\$ is set to “”, a new sequence number will be generated and returned thru H0\$.

```
0099 ENTER F0,H0$,H1$,H2$,H3$,H4$,H0,H1,H2,H3,H4,H5,H6,H7,H8,H9,H5$,
H99$[ALL],H99[ALL],H6$
```

Variable	Description
F0	Channel ARHix file is opened on
H0\$	If set prior to calling ARHITRAN.PUB, that sequence will always be used when writing a new record to ARHix. If set to “”, the sequence number generated will be returned to the calling program
H1\$,H2\$,...H99[ALL]	ARHixxx IOList variables (see the Accounts Receivable File Layouts)

Exit Points	Explanation
EXIT	ARHITRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_ARHI_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the NXTSEQ record in ARHix.

```
1510 RESERVE_ARHI_SEQ:  
1530 ENTER F0,SEQ$
```

Variable	Description
F0	Channel ARHIx file is opened on
SEQ\$	Next available sequence number in ARHIx

ARPRIOP.PUB

ARPRIOP.PUB is a public program that OPENS files and DIMensions the string templates used in invoice printing.

```
0025 ENTER FILE[ALL] , HEAD$, DATA$, O2$, T$, PLP, ONLINE, TRAN_LEN, HEAD, KITON, STAT$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel numbers
HEAD\$	String template for invoice header printing
DATA\$	String template for invoice line item printing
O2\$	Accounts Receivable options string
T\$	Application tables file name
PLP	Invoice form type flag: 0 - preprinted forms 1 - plain paper 2 - 8½-by-11-inch forms
ONLINE	Online invoice flag: 0 - batch invoice printing 1 - online invoice
TRAN_LEN	Length of the transaction number (set to 4)
HEAD	FILE[] array element for the ARTHxxx file
KITON	Unused in AR
STAT\$	Valid statuses to print (usually set to “15” in AR): 1 - invoice 5 - credit memo

Exit Points	Explanation
EXIT	ARPRIOP.PUB does not return an error.

ARRESTRT.PUB

ARRESTRT.PUB is a public program that checks for aborted posts and posts in process on another terminal.

```
0100 ENTER F0,F1,BATCH$,TERM$,MSG$,PICKS$,PARAM,STATUS
```

Variable	Description
F0	Channel number of the ARBTxxx file
F1	Channel number of the ARTMTxxx file; set to 0 if not opened
BATCH\$	Batch ID to check
TERM\$	Terminal ID to check or FILL(4) to check all terminals
MSG\$	Returns a message string if the check fails
PICKS\$	Returns the pick screen options to restart the post
PARAM	Task to perform: 0 - check for aborted post on the terminal 1 - check for aborted post on all terminals 2 - check for batch in use on another terminal 3 - change print status flag for the batch 4 - create and lock temporary print lock file
STATUS	Returns a value based on the batch status: 0 - batch is available 1 - aborted post detected 2 - batch posting on another terminal 3 - aborted print job detected 4 - batch printing on another terminal 5 - batch ID not found 6 - unable to create or lock temporary post or print file -99 - error updating print status

Exit Points	Explanation
EXIT	ARRESTRT.PUB does not return an error.

ARTRAN.PUB

ARTRAN.PUB is a public program that will generate, validate, and reserve a transaction number for the ARTHx file. The returned number should be used as the transaction number before writing a new record to the ARTHx file.

```
0100 ENTER FILE[ALL] , TRAN, CT1, CT2, TH0$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
TRAN	FILE[TRAN] index that the ARTDx file is opened on
CT1	FILE[CT1] index that the ARCTx file is opened on
CT2	FILE[CT2] index that the ARCTx file is opened on. This must be different than CT1
TH0\$	Batch/Transaction number from ARTHx. Should contain batch ID set by the calling program + “ ” or “0000”. The transaction is returned in the last 4 characters.

Exit Points	Explanation
EXIT	ARTRAN.PUB does not return any errors.

ARVWDOC.PUB

ARVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for customer document inquiry.

```
ENTER FILE[ALL],CUSTID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CUSTID\$	Customer ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	ARVWDOC.PUB does not return any errors.

BKAUDIT.PUB

BKAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Accounts Receivable module.

FILE_CHECK:

This entry point returns the total number of files in Bills of Materials that has GL Accounts in them.

```
1010 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	BKAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILES$[ALL]
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILES\$[ALL]	2-dimensional array dimensioned to DATAFILES\$[1:APPFILES,1:2] DATAFILES\$[x,1] = Name of file to be opened DATAFILES\$[x,2] = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Bills of Materials.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of BKAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

No data files contain GL Accounts in Bills of Materials.

The table that contain GL Accounts is the BKGLx table.

BKHIDRIL.PUB

BKHIDRIL.PUB is a public program that handles the drill down ability from General Ledger to Bills of Material detail history information. This program will build a scratch file containing matching historical entries between the GLDRx and BKHIx files, then call either BKCPNHST.PUB if the user is drilling on journal entries coming from Bills of Material.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	BKHIDRIL.PUB does not return any errors.

BKVWDOC.PUB

BKVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for kit and bills of material document inquiry.

```
ENTER FILE[ALL],BOMID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
BOMID\$	Kit or Bills of Materials ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	BKVWDOC.PUB does not return any errors.

BRAUDIT.PUB

BRAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Bank Reconciliation module.

FILE_CHECK:

This entry point returns the total number of files in Bank Reconciliation that has GL Accounts in them.

```
1010 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	BRAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Bank Reconciliation.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of BRAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: BRJRx and BRRAx.

No tables contain GL Accounts in Band Reconciliation

BRBAL.PUB

BRBAL.PUB is a public program that calculates the GL balance and unposted balance of GLJRx entries for a given GL account.

```
0100 ENTER GLBALANCE, FILE [ALL] , GLJR, O2$, JRNLTOT, G1 [ALL] , G0$, JRNL$, PDTHRU$,  
BANK_TYPE$
```

Variable	Description
GLBALANCE	Sum of GLMAx periods and GLJRx unposted entries for a specified GL account
FILE[ALL]	Standard file array used for the data file channel number
GLJR	FILE[GLJR] element that the GLJRx file is opened on
O2\$	Bank Reconciliation options
JRNLTOT	Only the total of GLJRx unposted entries for a specified GL account
G1[ALL]	From GLMAx file, already read into memory for current GL account
G0\$	Company ID + GL Account to get balances for
JRNL\$	Set to “NO” if ignoring GLJRx file balances
PDTHRU\$	Ending period to balance thru
BANK_TYPE\$	Set to C for credit card banks, any other value for normal banks

Exit Points	Explanation
EXIT	BRBAL.PUB does not return any errors.

BRTRAN.PUB

BRTRAN.PUB is a public program that writes Bank Reconciliation transactions to the BRTRxxx file.

```
0090 ENTER FO,TR0$,TR1$,TR2$,TR3$,TR4$,TR1,TR2,TR5$,TR6$,TR7$,TR99$[ALL],  
0090:TR99[ALL]
```

Variable	Description
FO	Channel number of the BRTRxxx file
TR0\$,...,TR99[ALL]	BRTRxxx IOList variables (see the Bank Reconciliation file layouts)

Exit Points	Explanation
EXIT	BRTRAN.PUB normally exits without an error.
EXIT ERR	An error is returned when an error occurs.

BRTRDRIL.PUB

BRTRDRIL.PUB is a public program that handles the drill down ability from General Ledger to Bank Reconciliation transaction information. This program will find the matching record between the GLDRx and BRTRx file, and display the transaction information for that record by calling BRDRVIEW.PUB.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	BRTRDRIL.PUB does not return any errors.

CJAUDIT.PUB

CJAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Contractors' Job Cost module.

FILE_CHECK:

This entry point returns the total number of files in Contractors' Job Cost that has GL Accounts in them.

```
1510 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	CJAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Contractors' Job Cost.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of CJAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are the CJBSx and CJWCx

The table that contain GL Accounts is the JOBGL table.

CJHIDRIL.PUB

CJHIDRIL.PUB is a public program that handles the drill down ability from General Ledger to Contractors' Job Cost detail history information. This program will build a scratch file containing matching historical entries between the GLDRx and CJHix files, then call CJIDHCON.PUB to display the matching records.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	CJHIDRIL.PUB does not return any errors.

CJHITRAN.PUB

ARHITRAN.PUB is a public program that handles writing records to the Contractors' Job Cost detail history file. This interface is called from routines that need to write new records to CJHix.

(Normal entry point)

This entry point is used when new records are to be written to the CJHix file. If JH0\$ is set prior to calling CJHITRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If JH0\$ is set to "", a new sequence number will be generated and returned thru JH0\$.

```
0099 ENTER F0,JH0$,JH1$,JHT$,JH2$,JH3$,JH4$,JH4 [ALL] ,JH5$,JH5,JH6,JH99$ [ALL] ,
JH99 [ALL]
```

Variable	Description
F0	Channel CJHix file is opened on
JH0\$	If set prior to calling CJHITRAN.PUB, that sequence will always be used when writing a new record to CJHix. If set to "", the sequence number generated will be returned to the calling program
JH1\$,...JH99[ALL]	CJHixxx IOList variables (see the Contractors' Job Cost File Layouts)

Exit Points	Explanation
EXIT	CJHITRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_CJHI_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the NXTSEQ record in CJHix.

```
3010 RESERVE_CJHI_SEQ:  
3020 ENTER F0,SEQ$
```

Variable	Description
F0	Channel CJHIx file is opened on
SEQ\$	Next available sequence number in CJHIx

CJOSCNVT.PUB

CJOSCNVT.PUB is a public program that will optionally convert data files from OSAS Job Cost version 7.5, to OSAS Contractors' Job Cost version 7.5. This function is designed to run from the F4 other commands menu, from the "Call a BASIC Program" function.

```
ENTER X0$,OSGUI$
```

Variable	Description
X0\$	Parameter pass thru the Call a BASIS Program function
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	CJOSCNVT.PUB does not return any errors.

CJVWDOC.PUB

CJVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for job and phase document inquiry.

```
ENTER FILE[ALL],JOBID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
JOBID\$	Job/Phase ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	CJVWDOC.PUB does not return any errors.

DDFOVCH.PUB

DDFOVCH.PUB is a public program that will generate a BBj PDF voucher form from properly formatted data files. This program will take the data and produce a Direct Deposit voucher looking similar to the OSAS laser form.

PRINT_FORM:

This entry point is used to print an invoice form after data is set up in temporary files.

```
0050 PRINT_FORM:
0060 ENTER VCHINFO$,DDVFH$,DDVFD$,FILE[ALL]
```

Variable	Description
VCHINFO\$	Template used for generic form information
DDVFH\$	Template for voucher header temporary file
DDVFD\$	Template for voucher detail temporary file
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	DDFOVCH.PUB normally exits with no error.

GET_TEMPLATES:

This entry point is to retrieve the template format that will be set up and used for general form information, as well as the temporary files built for use for printing vouchers.

```
4010 GET_TEMPLATES;
4020 ENTER VCHINFO$,DDVFH$,DDVFD$
```

Variable	Description
VCHINFO\$	Template used for generic form information
DDVFH\$	Template for voucher header temporary file
DDVFD\$	Template for voucher detail temporary file

The following chart breaks down the VCHINFO\$ template.

Variable	Description
PRINTNAME:C(30*)	BBj Java name of printer to print to
FORMID:C(8*)	Form ID from scratch files to print
OUTPATH:C(65*)	Path and name of PDF file to generate if e-mailing or printing to file
LOGO:C(80*)	Path and name of logo image to print on form
COPYTEXT:C(15*)	Text to print if printing multiple copies of form
INDEXHEAD:N(2*)	FILE[x] element that header scratch file is opened on (ARIFHxxx)
INDEXDETL:N(2*)	FILE[x] element that detail scratch file is opened on (ARIFDxxx)
STUBNAME:C(30*)	Title of stub
COMPNAME:C(30*)	Current company's name
COMPADD1:C(30*)	Current company's Address Line 1
COMPADD2:C(30*)	Current company's Address Line 2
COMPADD3:C(30*)	Current company's Address Line 3
COMPADD4:C(30*)	Current company's Address Line 4
BANKADD1:C(30*)	Bank's Address Line 1
BANKADD2:C(30*)	Bank's Address Line 2
BANKADD3:C(30*)	Bank's Address Line 3
BANKADD4:C(30*)	Bank's Address Line 4
BANKADD5:C(30*)	Bank's Address Line 5
LOGOP:C(2*)	L=Left justified, other values = Centered
FORMPROG:C(12*)	"DDFOVCH.PUB". Set in a variable so that the multi-copy form printing program knows what program to call to print invoices

See the Direct Deposit file layouts for DDVFHxxx and DDVFDxxx for layouts of the temporary files and their templates.

FAAUDIT.PUB

FAAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Fixed Assets module.

FILE_CHECK:

This entry point returns the total number of files in Fixed Assets that has GL Accounts in them.

```
1510 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	FAAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Fixed Assets.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of FAAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data file that contain GL accounts is the FAMAx file.

No tables contain GL accounts in Fixed Assets.

FACOMP.PUB

FACOMP.PUB is a public program that calculates an asset's period depreciation.

```
1010 ENTER C$,A1[ALL],A12$[ALL],A14$[ALL],ACPRD[ALL],D[ALL],D1$,DATEPLACED,
1010:FILE[ALL],CNVT$
```

Variable	Description
C\$	The depreciation columns to compute depreciation for (set each position to 1 to calculate that column or “ ” to skip that column): C\$(1,1) - Book column C\$(2,1) - FedTax column C\$(3,1) - Other column C\$(4,1) - A.M.T. column
A1[ALL],A12\$[ALL], A14\$[ALL]	FAMAxXX variables of the same name (see the Fixed Assets File Layouts)
ACPRD[ALL]	Fixed Assets Accounting Periods Table values
D[ALL]	Returns the calculated depreciation values: D[0,0] - Book depreciation (base method) D[0,1] - Book depreciation (switch method) D[1,0] - FedTax depreciation (base method) D[1,1] - FedTax depreciation (switch method) D[2,0] - Other depreciation (base method) D[2,1] - Other depreciation (switch method) D[3,0] - A.M.T. depreciation (base method) D[3,1] - A.M.T. depreciation (switch method)
D1\$	Returns the switch-required flag for each column (“Y” - switch is required, “N” - no switch): D1\$(1,1) - Book column D1\$(2,1) - FedTax column D1\$(3,1) - Other column D1\$(4,1) - A.M.T. column
DATEPLACED	Date the asset was placed in service (Julian value)
FILE[ALL]	Standard file array used for the data file channel numbers

Variable	Description
CNVT\$	Resource Manager Period Conversion table values (09\$)

Exit Points	Explanation
EXIT	FACOMP.PUB normally exits without an error.
EXIT 1	Returns 1 if the Period Conversion table is invalid.

FAVWDOC.PUB

FAVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for asset document inquiry.

```
ENTER FILE[ALL],ASSETID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
ASSETID\$	Asset ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	FAVWDOC.PUB does not return any errors.

GLACGFM.PUB

GLACGFM.PUB is a public program that handles **F6 Maintenance** for General Ledger accounts. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],MAINT$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
MAINT	GL Account entered by the user before pressing F6
O2\$	Options string (not used)
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	GLACGFM.PUB does not return any errors.

GLAUDIT.PUB

GLAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the General Ledger module.

FILE_CHECK:

This entry point returns the total number of files in General Ledger that has GL Accounts in them.

```
1010 FILE_CHECK:
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	GLAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:
1530 ENTER APPFILES, DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within General Ledger.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of GLAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: GLALx, GLJRx, GLMAx, and GLREx.

No tables in General Ledger contain GL Accounts.

GLJRNL.PUB

GLJRNL.PUB is a public program that writes General Ledger transactions to the GLJRxxx file.

(Normal entry point)

This entry point is used when new records are to be written to the GLJR_x file. If J0\$ is set prior to calling GLJRNL.PUB, the sequence number contained in that variable will always be used when writing a new record. If J0\$ is set to “”, a new sequence number will be generated and returned thru J0\$.

```
0090 ENTER F0,J1[ALL],J1$,J2$,J3$,J4$,J5$,J6$,J7$,J8$,J0$
```

Variable	Description
F0	Channel number of the GLJRxxx file
J1[ALL],...,J8\$	GLJRxxx IOList variables (see the General Ledger File Layouts)
J0\$	If set prior to calling GLJRNL.PUB, that sequence will always be used when writing a new record to GLJR _x . If set to “”, the sequence number generated will be returned to the calling program.

Exit Points	Explanation
EXIT	GLJRNL.PUB normally exits without an error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_GLJR_ENTRY:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the ENTRY record in GLJR_x.

```
1210 RESERVE_GLJR_ENTRY:
1220 ENTER F0,SEQ$
```

Variable	Description
F0	Channel number of the GLJRxxx file
SEQ\$	Sequence number returned to calling program

Exit Points	Explanation
EXIT	GLJRNL.PUB normally exits without an error.

GLYEARS.PUB

GLYEARS.PUB is a public program that builds a scratch file with active General Ledger financial years running in OSAS. This interface is called from both the GLSETUP.PUB function (F9 from the menu), which allows users to switch to prior year files, and from Change Fields, allowing users to change General Ledger key fields across multiple years of General Ledger files.

(Normal entry point)

This entry point is used to build an m-keyed file that contains a list of General Ledger years. This is done by looking for the existence of the MASK\$ field in file names when searching thru the current dataset.

```
0100 ENTER F$,F0,MASK$,FILE[ALL]
```

Variable	Description
F\$	Name of m-keyed file built that contains GL years
F0	Channel F\$ is opened on
MASK\$	Mask field to use to search thru dataset (ex: "GLMAH.Y??", "GLMAH.Y*"). ? = Any single character * = Any number of characters
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	GLYEARS.PUB exits with no error.

CUSTOM_SCRATCH_FILE:

This entry point is used identically to how the normal entry point is used, with the exception that the scratch file built will contain 2 fields per record: the GL year and the word "NO ". This is used by the Change Fields function to build a list of years available for changing fields for.

```
0130 CUSTOM_SCRATCH_FILE:
0140 ENTER F$,F0,MASK$,FILE[ALL]
```

Variable	Description
F\$	Name of m-keyed file built that contains GL years
F0	Channel F\$ is opened on
MASK\$	Mask field to use to search thru dataset (ex: “GLMAH.Y??”, “GLMAH.Y*”). ? = Any single character * = Any number of characters
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	GLYEARS.PUB exits with no error.

INACCLFM.PUB

INACCLFM.PUB is a public program that handles **F6 Maintenance** for customer levels. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Customer level entered by the user before pressing F6
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INACCLFM.PUB does not return any errors.

INACGLFM.PUB

INACGLFM.PUB is a public program that handles **F6 Maintenance** for Inventory General Ledger codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	General Ledger account code entered by the user before pressing F6
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INACGLFM.PUB does not return any errors.

INACIFM.PUB

INACIFM.PUB is a public program that handles **F6 Maintenance** for items. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE [ALL] , CODE$ , O2$ , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Item ID entered by the user before pressing F6
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INACIFM.PUB does not return any errors.

INACLFM.PUB

INACLFM.PUB is a public program that handles **F6 Maintenance** for locations. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
0045 ENTER FILE[ALL] , CODE$, O2$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Location ID entered by the user before pressing F6
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INACLFM.PUB does not return any errors.

INACSCFM.PUB

INACSCFM.PUB is a public program that handles **F6 Maintenance** for sales categories. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
0045 ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Sales category entered by the user before pressing F6
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INACSCFM.PUB does not return any errors.

INACUFM.PUB

INACUFM.PUB is a public program that handles **F6 Maintenance** for units of measure. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
0045 ENTER FILE[ALL],CODE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Unit of measure entered by the user before pressing F6
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INACUFM.PUB does not return any errors.

INAUDIT.PUB

INAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Inventory module.

FILE_CHECK:

This entry point returns the total number of files in Inventory that has GL Accounts in them.

```
1510 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	INAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Inventory.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of INAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: INGLx and INTRx.

No tables in Inventory contain GL accounts.

INBLDQTY.PUB

INBLDQTY.PUB is a public program that creates a scratch file from the Inventory Transactions file for use by the Inventory **Rebuild Item Quantities** function.

```
0060 ENTER FILE[ALL] , PASS$ , INQT$ , INSN$ , INVE$ , INLD$ , INLS$ , MSG$ , TASK , OPTION$
```

Variables	Description
FILE[ALL]	Standard file array used for the file channel number
PASS\$	Template array used for the data file channel number
INQT\$	INQTxxx file template (Quantity Totals)
INSN\$	INSNxxx file template (Serial Numbers)
INVE\$	INVExxx file template (Item Master)
INLD\$	INLDxxx file template (Location Detail)
INLS\$	INLSxxx file template (Transaction Lot/Serial Number)
MSG\$	Error messages returned to calling program
TASK	0 = Open files and create sort file; 1 = Open files Only
OPTION\$	Application specific options and interfaces string

Exit Points	Explanation
EXIT	INBLDQTY.PUB does not return any errors.

The following chart breaks down the PASS\$ template.

Field	Description
INVE:N(2*)	Channel number on which the INVEx file is open
INLD:N(2*)	Channel number on which the INLDx file is open
QTSCR:N(2*)	Channel on which the INQTxxx scratch file is open
SNSCR:N(2*)	Channel on which the INSNxxx scratch file is open
ERRSCR:N(2*)	Channel on which the SCRT3.zzz error scratch file is open

INCALCPR.PUB

INCALCPR.PUB is a public program that calculates a sales price based on item, quantity and customer criteria. It is called by INUPD.PUB, INPRCALC.PUB and INQTYDSP.PUB.

```
0045 ENTER FILE[ALL], PCALC$, INUPD3$, MSG$, INVE$, INUM$, INLD$, INPS$, INLP$, INUP$,
0045: INSN$, INPP$, O2$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
PCALC\$	INCALCPR.PUB data string (see table below)
INUPD3\$	File array template (see INUPD.PUB)
MSG\$	Returns a message string if an error occurs
INVE\$	INVExxx file template (Item Master file)
INUM\$	INUMxxx file template (Unit of Measure file)
INLD\$	INLDxxx file template (Location Detail file)
INPS\$	INPSxxx file template (Price Structures file)
INUP\$	INUPxxx file template (Unit of Measure Pricing file)
INSN\$	INSNxxx file template (Serial Number file)
INPP\$	INPPxxx file template (Promotional Pricing file)
O2\$	Inventory options string

Exit Points	Explanation
EXIT	INCALCPR.PUB does not return any errors.

PCALC\$ String Template Definition

The PCALC\$ string template is used to pass information to the INCALCPR.PUB program. The fields in the template are defined in the following table. If INCALCPR.PUB is CALLED with PCALC\$ set to “”, the program will DIMension PCALC\$ using the template and EXIT back to the CALLing program.

Field	Type	Len	Description
ITEM\$	C	20	Item being sold
SERN\$	C	35	If a serial number is specified, the program calculates the price for that serial number; if not, the program calculates a generic item price
LOCA\$	C	6	Item location ID for the sale
PRID\$	C	6	Price ID to be used in the calculation
CUST\$	C	6	Customer ID (optional)
CLVL\$	C	6	Customer Level
UNIT\$	C	5	Unit of measure for the sale
SQTY	B	8	Quantity to be sold
SDAT	N	7	Date of the sale
PROMO	C	6	Returns the promotional price ID used in the calculation
UPRC	B	8	Returns the unit price of the item
EPRC	B	8	Returns the extended price of the item
OPRC	B	8	Returns the item's unit price without the promotional price applied
MPRC	B	8	Returns the minimum price for the item

INCMNT.PUB

INCMNT.PUB is a public program that provides entry into GENCMNT.PUB for item comment inquiry and maintenance.

```
ENTER FILE[ALL],ITEMID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
ITEMID\$	Item ID for which comments are displayed and edited
X0\$	A = Maintenance mode; B = Inquiry only
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INCMNT.PUB does not return any errors.

INHCDRIL.PUB

INHCDRIL.PUB is a public program that handles the drill down ability from General Ledger to Inventory COGS history information. This program will build a scratch file containing matching historical entries between the GLDRx and INHCx files, then call INICHCON to display the COGS entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	INHCDRIL.PUB does not return any errors.

INHCTRAN.PUB

INHCTRAN.PUB is a public program that handles writing records to the Inventory GL adjustments history file. This interface is called from routines that need to write new records to INHCx.

(Normal entry point)

This entry point is used when new records are to be written to the INHCx file. If INHC.SEQ\$ is set prior to calling INHCTRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If INHC.SEQ\$ is set to "", a new sequence number will be generated and returned thru INHC.SEQ\$

```
0099 ENTER F0,INHC$
```

Variable	Description
F0	Channel INHCx file is opened on
INHC\$	INHCx string template (see the Inventory File Layouts)

Exit Points	Explanation
EXIT	INHCTRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_INHC_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the INHC.SEQ\$ variable. The next available number is then written back to the NXTSEQ record in INHCx.

```
3010 RESERVE_INHC_SEQ:  
3020 ENTER F0,INHC$
```

Variable	Description
F0	Channel APHIX file is opened on
INHC\$	INHCx string template (see the Inventory File Layouts)

INHIDRIL.PUB

INHIDRIL.PUB is a public program that handles the drill down ability from General Ledger to Inventory detail history information. This program will build a scratch file containing matching historical entries between the GLDRx and INHlx files, then call INIDHCON to display the detail history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL], OSGUI$, GLJRSEQ$, YEAR_EXT$, MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	INHIDRIL.PUB does not return any errors.

INITEM.PUB

INITEM.PUB is a public program that handles item ID input and corresponding item lookup functions. Use INITEM.PUB anywhere that an item ID is entered. You can control the way the entry program functions with the INGET\$ variable.

```
0060 ENTER FILE[ALL] , INVE$, INLD$, INGET$, Y$, O2$, OSGUI$
```

Variables	Description
FILE[ALL]	Standard file array used for the data file channel number
INVE\$	INVExxx file template (Item Master file)
INLD\$	INLDxxx file template (Location Detail file)
INGET\$	INITEM data template (see table below)
Y\$	Standard GENINPUT command string
O2\$	Inventory options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INITEM.PUB does not return any errors.

INGET\$ String Template Definition

The INGET\$ string template is used to pass information to the INITEM.PUB program. The fields in the string template are defined in the following table.

Field	Type	Len	Description
APP\$	C	2	Application calling INITEM.PUB. Used for determining whether to display customer or vendor alternates in the Customer/Vendor lookup. "IN" will allow both lookups

Field	Type	Len	Description
FORCE\$	C	1	Valid Item ID flag Y - force the user to enter a valid item ID N - allow an invalid item ID
BLANK\$	C	1	Blank Item ID flag Y - allow a blank item ID N - prevent a blank item ID
LOCA\$	C	6	Current location ID of the line item. If this value is set, INITEM.PUB will return the INLD\$ template to the calling program
SRCE\$	C	6	Current vendor or customer of the line item
SUP\$	C	1	Supersede flag Y - check for supersedes N - do not check for supersedes
REP\$	C	1	Pick screen entry flag Y - report pick screen item ID prompt N - normal item ID entry
FOUND\$	C	1	Returned to calling program: Y - the item was found N - the item was not found
PROG\$	C	1	Maintenance flag Y - allow F6 maintenance N - do not allow F6 maintenance
MSG\$	C	1	Warning on Item ID change flag Y - print warning message if item is changed N - do not print a warning message

Field	Type	Len	Description
SVOK\$	C	1	Service item flag Y - allow service items N - do not allow service items
TTYP\$	C	2	Transaction type; should be the same as INUPD1.TTYP\$
ALT\$	C	20	The alternate item ID that was entered by the user or set by one of the item searches
YDIS\$	C	10	GENINPUT.PUB's YDISABLE\$
STAT\$	C	5	List of valid item or item/location statuses for the function (for example, "ADO" allows active, discontinued and obsolete items only): A - active D - discontinued S - superseded O - obsolete K - kits
WND\$	C	12	Override F2 Inquiry window ID
SLV\$	C	35	Override F2 Inquiry SELVAL\$ value
X0	N	2	GENINPUT.PUB's X0 screen position
X1	N	2	GENINPUT.PUB's X1 screen position
FLD_HLP\$	C	12	Field Help value for F1 help
GUI_WIND	N	5	Channel on which the GUI device X0 is open
GUI_ID	N	5	Control ID of Item ID

INQTYDSP.PUB

INQTYDSP.PUB is a public program that displays the available quantities for any item location and the quantity breaks for the current unit of measure. It is designed for optional use with the AR and SO entry programs.

```
0045 ENTER FILE[ALL] , INUPD1$ , INUPD3$ , INLD$ , PRID$ , CUST$ , CLVL$ , UNIT$ , SDAT , X9$ ,
0045 :DISPFLAG , INVE$ , INUM$ , INPS$ , INLP$ , INUP$ , INSN$ , INPP$ , O2$ , OSGUI$ ,
0045 : INQTYDSP_CONTEXT
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
INUPD1\$	INUPD current-values template (see INUPD.PUB)
INUPD3\$	File array template (see INUPD.PUB)
INLD\$	INLDxxx file template (Location Detail file)
PRID\$	Price ID
CUST\$	Customer ID
CLVL\$	Customer level
UNIT\$	Unit of measure
SDAT	Transaction sale date
X9\$	Returns error message string if an error occurs
DISPFLAG	Information Display flag: 0 - Clear display 1 - Display item information 2 - Display “Not Applicable” for noninventory items
INVE\$	INVExxx file template (Item Master file)
INUM\$	INUMxxx file template (Units of Measure file)
INPS\$	INPSxxx file template (Price Structures file)
INLP\$	INLPxxx file template (Location Pricing file)
INUP\$	INUPxxx file template (Units of Measure Pricing file)
INSN\$	INSNxxx file template (Serial Number file)
INPP\$	INPPxxx file template (Promotional Pricing file)

Variable	Description
O2\$	Inventory options string
OSGUI\$	Template for GUI functions
INQTYDSP_CONTEXT	Context in which the graphical screen exists

Exit Points	Explanation
EXIT	INQTYDSP.PUB does not return any errors.

INSERTLOT.PUB

INSERTLOT.PUB is a public program that provides lot and/or serial number entry and updates Inventory files appropriately. To use INSERTLOT.PUB for lot/serial entry, your application must use a transaction lot/serial number file identical in layout to the INLSxxx file.

```
0060 ENTER FILE[ALL],PASS$,INLS$,INUPD1$,INUPD2$,INUPD3$,INCJ$,INLD$,INHI$,
0060:INH$,INLH$,INLT$,INQL$,INQT$,INSH$,INSN$,INUM$,O2$,PASS_TASK$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
PASS\$	INSERTLOT data template (see table below)
INLS\$	INLSxxx file template (Transaction Lot/Serial Number file)
INUPD1\$	INUPD current-values template (see INUPD.PUB)
INUPD2\$	INUPD old-values template (see INUPD.PUB)
INUPD3\$	File array template (see INUPD.PUB)
INCJ\$	INCJxxx file template (GL Adjustments file)
INLD\$	INLDxxx file template (Location Detail file)
INHI\$	INHxxx file template (Detail History file)
INH\$	INH\$xxx file template (Summary History file)
INLH\$	INLHxxx file template (Lot History file)
INLT\$	INLTxxx file template (Lot Numbers file)
INQL\$	INQLxxx file template (Quantity Locations file)
INQT\$	INQTxxx file template (Quantity Totals file)
INSH\$	INSHxxx file template (Serial History file)
INSN\$	INSNxxx file template (Serial Numbers file)
INUM\$	INUMxxx file template (Units of Measure file)
O2\$	Inventory options string

Variable	Description
PASS_TASK\$	Source transaction task: A - appending a line E - editing a line
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	INSERTLOT.PUB does not return any errors.

PASS\$ String Template Definition

The PASS\$ string template is used to pass information to INSERTLOT.PUB. The template is defined as follows:

Field	Type	Len	Description
IDESS	C	35	Description of item
LDESS	C	30	Description of location
TYPE\$	C	1	Entry type: 1 - serial entry 2 - lot entry 3 - serial/lotted entry
TQTY1	B	8	Total ordered quantity
TQTY2	B	8	Total shipped/received quantity
TQTY3	B	8	Total backordered quantity
TCST	B	8	Total cost of all lots or serial numbers
TPRC	B	8	Total price of all serial numbers

INSERT2.PUB

INSERT2.PUB is a public program that provides lot and/or serial number entry and updates Inventory files appropriately. It is used instead of INSERT.PUB when the user selects the option to enter the lot number first and then the serial number. The variables used are identical. See INSERT.PUB for a description of the variables used.

INSERROL.PUB

INSERROL.PUB is a public program that rolls the serial numbers in the xxLSxxx file for line inserts.

```
0100 ENTER FILE[ALL] , APP$ , BATCH$ , TRAN$ , LINE$ , ROL_LOC[ALL] , CHANNEL
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
APP\$	Application ID
BATCH\$	Batch ID of the transaction (FILL(6) for AP and PO)
TRAN\$	Transaction ID
LINE\$	Line number to insert
ROL_LOC[ALL]	GENLINE LOC[] array
CHANNEL	FILE[] array index for the xxLSxxx file

Exit Points	Explanation
EXIT	INSERROL.PUB does not return an error.

INUPD

INUPD is a general routine that calls INUPD.PUB to update the Inventory files for a transaction. INUPD occupies lines 8500–8540. Merge it into a program only if the program must update Inventory files.

Variable	Description
INUPDERR	Returns 0 if successful or 1 if an error occurs.

INUPD.PUB

INUPD.PUB is a public program that updates Inventory files for a transaction. INUPD.PUB should be called only through the INUPD routine.

```
0058 ENTER FILE[ALL] , INUPD1$ , INUPD2$ , INUPD3$ , INCJ$ , INLD$ , INHI$ , INHS$ , INLH$ ,
0058 : INLP$ , INLT$ , INPP$ , INPS$ , INQL$ , INQT$ , INSH$ , INSN$ , INUM$ , INUP$ , INVE$ , O2$ , MSG$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
INUPD1\$	INUPD current-values template (see table below)
INUPD2\$	INUPD old-values template (see table below)
INUPD3\$	File array template for INUPD.PUB (see table below)
INCJ\$	INCJxxx file template (GL Adjustments file)
INLD\$	INLDxxx file template (Location Detail file)
INHI\$	INHIxxx file template (Detail History file)
INHS\$	INHSxxx file template (Summary History file)
INLH\$	INLHxxx file template (Lot History file)
INLP\$	INLPxxx file template (Location Pricing file)
INLT\$	INLTxxx file template (Lot Numbers file)
INPP\$	INPPxxx file template (Promotional Pricing file)
INPS\$	INPSxxx file template (Price Structure file)
INQL\$	INQLxxx file template (Quantity Locations file)
INQT\$	INQTxxx file template (Quantity Totals file)
INSH\$	INSHxxx file template (Serial History file)
INSN\$	INSNxxx file template (Serial Numbers file)
INUM\$	INUMxxx file template (Units of Measure file)
INUP\$	INUPxxx file template (Unit Pricing file)
INVE\$	INVExxx file template (Items file)
O2\$	Inventory options string
MSG\$	Returns a message string if an error occurs

Exit Points	Explanation
EXIT	INUPD.PUB normally exits without an error.
EXIT ERR	An error is returned when an error occurs.

The INUPDSET.PUB program DIMensions and assigns some of the variables in the INUPD1\$, INUPD2\$ and INUPD3\$ strings.

INUPD1\$ and INUPD2\$ String Template Definition

The INUPD1\$ and INUPD2\$ string templates are identical. The values contained in the string represent the new and previous values (if any) for the transaction. The template contains the fields defined in the following table:

Field	Type	Len	Description
TASK\$	C	2	Update task: P1 - PO new order P2 - PO receive goods P3 - PO return goods P4 - AP invoice P5 - AP debit memo S1 - SO new order S2 - SO shipped order S3 - SO verify order S4 - AR invoice S5 - AR credit memo S6 - AR/SO post invoice S7 - AR/SO post credit memo T1 - AP material requisition T2 - AP return material requisition T3 - Inventory transfer from T4 - Inventory transfer to B1 - Build from B2 - Build to B3 - Unbuild from B4 - Unbuild to A1 - Quantity adjustment increase A2 - Quantity adjustment decrease UC - Unit conversion QT - Get INQTxxx record CT - Get INLDxxx cost (locks record) CS - Get cost of sale PR - Get price of sale TC - Get cost of transfer WT - Calculate weight of item

Field	Type	Len	Description
ITEM	C	20	Item ID
LOCA	C	6	Location ID
LOTN	C	16	Lot number
SERN	C	35	Serial number
GLCD	C	2	GL account code
CMNT	C	35	Comment (serial numbers)
PRID	C	6	Price ID
CLVL	C	6	Customer level
CUST	C	6	Customer ID
UNIT	C	5	Unit of measure
SREP	C	3	Sales rep
PENT	N	1	Penalty type
ODAT	N	7	Order date
IVDAT	N	7	Invoice date
RDAT	N	7	Receipt date
TPER	N	2	Transaction period
TYR	N	4	Transaction year
PROM	C	6	Promotion used in price calculation
HISTFL	N	1	Update history flag (set to 1 for update or 0 for no update)
INHISQ	C	6	INHlxxx sequence number
INLHSQ	C	6	INLHxxx sequence number
INSHSQ	C	6	INSHxxx sequence number
APP	C	2	Application ID
BATCH	C	6	Batch ID
TRAN	C	8	Transaction ID
LINE	C	3	Line number
SRCE	C	6	Source ID

Field	Type	Len	Description
SREF	C	8	Reference ID
JOB	C	6	Job ID
PHASE	C	6	Phase ID
CCODE	C	3	Cost code
ITYP	C	1	Item type
LOTS	N	1	1 if item is lotted or 0 if not lotted
KITS	N	1	1 if item is a kit or 0 if not a kit
DEL	N	1	1 if deleting a line or 0 if not
UCST	B	8	Unit cost, in base unit of measure
ECST	B	8	Extended cost
TCOST	B	8	Transfer cost
SUCST	B	8	Selling unit cost (returned)
SECST	B	8	Selling extended cost (returned)
CONV	B	8	Conversion factor
PENA	B	8	Penalty amount
OQTY1	B	8	Original quantity (in selling or purchasing units)
OQTY2	B	8	Shipped/received quantity (in selling or purchasing units)
OQTY3	B	8	Backordered quantity (in selling or purchasing units)
BQTY1	B	8	Original quantity (in base units)
BQTY2	B	8	Shipped/received quantity (in base units)
BQTY3	B	8	Backordered quantity (in base units)
QTQTY1	B	8	On-hand quantity from INQTxxx
QTQTY2	B	8	Committed quantity from INQTxxx
QTQTY3	B	8	In-use quantity from INQTxxx
QTQTY4	B	8	On-order quantity from INQTxxx
QTQTY5	B	8	Available quantity (on-hand - in-use - committed)
PRC1	B	8	Unit price

Field	Type	Len	Description
PRC2	B	8	Extended price
PRC3	B	8	Calculated price before promo
PRC4	B	8	Minimum price
CST1	B	8	Standard cost from INLDxxx
CST2	B	8	Average cost from INLDxxx
CST3	B	8	Base cost from INLDxxx
CST4	B	8	Last cost from INLDxxx
VAL1	B	8	Total cost of item from INQTxxx
VCOST	B	8	Unused
KIT2	N	1	Component of a kit. Flagged so average price on a component is not recalculated, because pricing is done on the kit level.
OJOB	C	1	Flags if item belongs with a PO with a job ID

INUPD3\$ String Template Definition

The INUPD3\$ string template is defined below. The values in the string represent information about the FILE[] array elements for various Inventory data files.

Field	Type	Len	Description
FLCJ	N	2	File[x] value for INCJxxx file (GL Adjustments file)
FLLD	N	2	File[x] value for INLDxxx file (Location Detail file)
FLHI	N	2	File[x] value for INHIxxx file (Detail History file)
FLHS	N	2	File[x] value for INHSxxx file (Summary History file)
FLLH	N	2	File[x] value for INLHxxx file (Lot History file)
FLLP	N	2	File[x] value for INLPxxx file (Location Pricing file)
FLLS	N	2	File[x] value for INLSxxx file (Lot/Serial Transactions file)
FLLT	N	2	File[x] value for INLTxxx file (Lot Numbers file)

Field	Type	Len	Description
FLPP	N	2	File[x] value for INPPxxx file (Promotional Pricing file)
FLPS	N	2	File[x] value for INPSxxx file (Price Structures file)
FLQL	N	2	File[x] value for INQLxxx file (Quantity Detail file)
FLQT1	N	2	File[x] value for INQTxxx file (Quantity Totals file).
FLSH	N	2	File[x] value for INSHxxx file (Serial History file)
FLSN	N	2	File[x] value for INSNxxx file (Serial Numbers file)
FLUM	N	2	File[x] value for INUMxxx file (Unit of Measure file)
FLUP	N	2	File[x] value for INUPxxx file (Unit Pricing file)
FLVE	N	2	File[x] value for INVExxx file (Items file)
FLVI	N	2	File[x] value for INVxxx file (Vendor Information file)

INUPDSET.PUB

INUPDSET.PUB is a public program that DIMensions the string templates used to work with INUPD. The program should be called in the *INI program in transaction programs or in the pick screen program of posts that interface to Inventory.

```
0058 ENTER FILE[ALL] , INUPD1$, INUPD2$, INUPD3$, FILE$, MSG$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
INUPD1\$	INUPD current-values string to be DIMensioned
INUPD2\$	INUPD old-values string to be DIMensioned
INUPD3\$	INUPD file-array string to be DIMensioned
FILE\$	xxLSxxx file name (Transaction Lot/Serial Number file)
MSG\$	Returns a message string if an error occurs

Exit Points	Explanation
EXIT	INUPDSET.PUB normally exits without an error.
EXIT ERR	An error is returned when an error occurs.

INVWDOC.PUB

INVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for item document inquiry.

```
ENTER FILE[ALL],ITEMID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
ITEMID\$	Item ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	INVWDOC.PUB does not return any errors.

JOACCFM.PUB

JOACCFM.PUB is a public program that handles **F6 Maintenance** for cost codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CC0$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CC0\$	Cost code entered by the user before pressing F6
O2\$	Job Cost options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	JOACCFM.PUB does not return any errors.

JOACJCFM.PUB

JOACJCFM.PUB is a public program that handles **F6 Maintenance** for Job/Phase cost codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],PASS$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
PASS\$	Cost code entered by the user before pressing F6
O2\$	Job Cost options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	JOACJCFM.PUB does not return any errors.

JOACJFM.PUB

JOACJFM.PUB is a public program that handles **F6 Maintenance** for jobs and phases. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],PASS$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
PASS\$	Job/Phase ID entered by the user before pressing F6
O2\$	Job Cost options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	JOACJFM.PUB does not return any errors.

JOACTFM.PUB

JOACTFM.PUB is a public program that handles **F6 Maintenance** for cost types. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CT0$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CT0\$	Cost type entered by the user before pressing F6
O2\$	Job Cost options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	JOACTFM.PUB does not return any errors.

JOAUDIT.PUB

JOAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Job Cost module.

FILE_CHECK:

This entry point returns the total number of files in Job Cost that has GL Accounts in them.

```
1510 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	JOAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Job Cost.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of JOAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data file that contain GL accounts is the JOBSx file.

The table that contain GL Accounts is the JOBGL table.

JOHIDRIL.PUB

JOHIDRIL.PUB is a public program that handles the drill down ability from General Ledger to Job Cost detail history information. This program will build a scratch file containing matching historical entries between the GLDRx and JOHix files, then call JOIDHCON.PUB to display the detail history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	JOHIDRIL.PUB does not return any errors.

JOHITRAN.PUB

JOHITRAN.PUB is a public program that handles writing records to the Job Cost detail history file. This interface is called from routines that need to write new records to JOHIX.

(Normal entry point)

This entry point is used when new records are to be written to the JOHIX file. If JH0\$ is set prior to calling JOHITRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If JH0\$ is set to “”, a new sequence number will be generated and returned thru JH0\$.

```
0099 ENTER F0,JH0$,JH1$,JHT$,JH2$,JH3$,JH4$,JH4 [ALL] ,JH5$,JH5,JH6,JH7,  
JH99$ [ALL] ,JH99 [ALL]
```

Variable	Description
F0	Channel JOHIX file is opened on
JH0\$	If set prior to calling JOHITRAN.PUB, that sequence will always be used when writing a new record to JOHIX. If set to “”, the sequence number generated will be returned to the calling program
JH1\$...JH99[ALL]	JOHIXxx IOList variables (see the Accounts Receivable File Layouts)

Exit Points	Explanation
EXIT	JOHITRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_JOHI_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the NXTSEQ record in JOHIX.

```
3010 RESERVE_JOHI_SEQ:  
3020 ENTER F0,SEQ$
```

Variable	Description
F0	Channel JOHIX file is opened on
SEQ\$	Next available sequence number in JOHIX

JOVWDOC.PUB

JOVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for job/phase document inquiry.

```
ENTER FILE[ALL],JOBID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
JOBID\$	Job/Phase ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	JOVWDOC.PUB does not return any errors.

LCAUDIT.PUB

LCAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Landed Cost module.

FILE_CHECK:

This entry point returns the total number of files in Landed Cost that has GL Accounts in them.

```
1010 FILE_CHECK:
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	LCAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:
1530 ENTER APPFILES, DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Landed Cost.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of LCAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data file that contain GL accounts is the LCLDx file.

The tables that contain GL Accounts are: APGL, DF, DM, and MRGL.

PAACEFM.PUB

PAACEFM.PUB is a public program that handles **F6 Maintenance** for employees. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],EMPLOYEE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
EMPLOYEE\$	Employee ID entered by the user before pressing F6
O2\$	Payroll options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	PAACEFM.PUB does not return any errors.

PAACTGFM.PUB

PAACTGFM.PUB is a public program that handles **F6 Maintenance** for tax groups. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],EMPLOYEE$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Tax Group ID entered by the user before pressing F6
O2\$	Payroll options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	PAACTGFM.PUB does not return any errors.

PAAUDIT.PUB

PAAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Payroll module.

FILE_CHECK:

This entry point returns the total number of files in Payroll that has GL Accounts in them.

```
1510 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	PAAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Payroll.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of PAAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAX exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAX file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAX file, as well as the accounts status

The data files that contain GL accounts are: PADDx, PADPx, PAECx, and PAWIx

The tables that contain GL Accounts are: GLDEP and GLPAY.

PACJCCFM.PUB

PACJCCFM.PUB is a public program that handles **F6 Maintenance** for class codes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL],CL$,O2$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CL1\$	Class code entered by the user before pressing F6
O2\$	Payroll or Direct Deposit options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	PACJCCFM.PUB does not return any errors.

PACJLIB.PUB

PACJLIB.PUB is a public program that handles a variety of Contractors' Job Cost functions from Payroll and Direct Deposit, via multiple entry points.

WIP_RANGE:

This entry point returns will read the JOBGL table from the CJTB file and return the WIP account range.

```
1005 WIP_RANGE:
1010 ENTER FILE[ALL], WIP1$, WIP2$, CJWIPDF$, CJTB_INDEX
```

Variable	Description
FILE[ALL]	Standard file array
WIP1\$	WIP Account from value
WIP2\$	WIP Account thru value
CJWIPDF\$	Default WIP account
CJTB_INDEX	FILE[x] index that CJTB is opened on

Exit Points	Explanation
EXIT	PACJLIB.PUB exits with no error.

CHECK_GL_WIP

This entry point will check the GL Account you entered against the WIP range and let the user know if the data is valid.

```
1105 CHECK_GL_WIP:
1110 ENTER FILE[ALL], OK$, WIP1$, WIP2$, GL_ACCT$, JOB_ID$, RET$, MSG$
```

Variable	Description
FILE[ALL]	Standard File Array
OK\$	Returns "Y" if GL Account is valid, blank if not valid
WIP1\$	WIP account from

Variable	Description
WIP2\$	WIP account thru
GL_ACCT\$	GL Account entered by the user to test
JOB_ID\$	Job ID entered by the user
RET\$	Retainage GL account from the APGL table
MSG\$	Returns an error message if GL account entered was not valid

ADD_MAN_TOTAL:

This entry point will accumulate manual check totals into the PABDx file.

```
1205 ADD_MAN_TOTAL:
1210 ENTER FILE[ALL] , PABD_INDEX, CW5$, SCREENPOS, LOC[ALL] , CH1$, TMP1[ALL] , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
PABD_INDEX	FILE[x] index that APINx is opened on
CW5\$	Employer paid withholdings flag - 'Y' or 'N'
SCREENPOS	Set to 1 if in the withholdings HLE
LOC[ALL]	Array used in HLE programs
CH1\$	Employee ID entered by the user
TMP1[ALL]	Withholding amounts
MSG\$	Message to display if invoice was not a retainage invoice

GET_GL_WIP:

This entry point will get the WIP GL account if Job/Phases are using specific WIP codes.

```
1305 GET_GL_WIP:
1310 ENTER FILE[ALL] , GL_ACCT$, JOB_ID$, B99$[ALL] , CJWC_INDEX
```

Variable	Description
FILE[ALL]	Standard file array
GL_ACCT\$	GL WIP account from the CJWCx file
JOB_ID\$	Job/Phase ID entered by the user
B99\$[ALL]	WIP information for the Job/Phase ID entered
CJWC_INDEX	FILE[x] index the CJWCx file is opened on

JOB_FROM:

This entry point displays an entry prompt for Job ID from.

```
1405 JOB_FROM:
1410 ENTER FILE[ALL],LISTYN$,JOB_FROM$,JOB_X0,JOB_X1,JOB_GUI,GUI,OSGUI$,Y$
```

Variable	Description
FILE[ALL]	Standard file array
LISTYN\$	“Y” = Display message ‘Enter ‘*’ to input a list of jobs’
JOB_FROM\$	Job ID entered by the user
JOB_X0	X0 position of the Job From field used by GENINPUT
JOB_X1	X1 position of the Job Thru field used by GENINPUT
JOB_GUI	GUI ID of the Job From prompt used by GENINPUT
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template
Y\$	Command returned by GENINPUT.PUB

JOB_THRU:

This entry displays an entry prompt for Job ID thru.

```
1505 JOB_THRU:
1510 ENTER FILE[ALL],JOB_FROM$,JOB_X0,JOB_X1,JOB_GUI,JOB_THRU$,GUI,OSGUI$,Y$
```

Variable	Description
FILE[ALL]	Standard file array
JOB_FROM\$	Job From ID entered previously entered by the user
JOB_X0	X0 position of the Job From field used by GENINPUT
JOB_X1	X1 position of the Job Thru field used by GENINPUT
JOB_GUI	GUI ID of the Job From prompt used by GENINPUT
JOB_THRU\$	Job ID entered by the user
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template
Y\$	Command returned by GENINPUT.PUB

SKILL:

This entry point will prompt for a Skill Level.

```
1605 CLEAR_LIST:
1610 ENTER FILE[ALL], PACL_INDEX, CL5[ALL], CE6$, CL1$, CL2$, SKILL$, SAVE_SKILL$,
SK$, CE99$[ALL], SKILL_X0, SKILL_X1, SKILL_GUI, GUI, OSGUI$, Y$
```

Variable	Description
FILE[ALL]	Standard file array
PACL_INDEX	FILE[x] element that PACLx is open on
CL5[ALL]	Pay rate skills
CE6\$	Labor Class
CL1\$	Class Code
CL2\$	Class Description
SKILL\$	List of skills
SAVE_SKILL\$	Original skill level entered by the user
SK\$	Skill level entered by the user
CE99\$[ALL]	Employee and record type
SKILL_X0	X0 position of the Skill Level prompt for GENINPUT
SKILL_X1	X1 position of the Skill Level prompt for GENINPUT

Variable	Description
SKILL_GUI	GUI ID of the Skill Level prompt for GENINPUT
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template
Y\$	Command returned from GENINPUT

UNION:

This entry point will paint and prompt for a Union Code.

```
1705 UNION:
1710 ENTER FILE [ALL] , UNION_X0 , UNION_X1 , UNION_GUI , CE99$ [ALL] , PAUN_INDEX , Y$ ,
GUI , OSGUI
```

Variable	Description
FILE[ALL]	Standard file array
UNION_X0	X0 position of the Union prompt for GENINPUT
UNION_X1	X1 position of the Union prompt for GENINPUT
UNION_GUI	GUI ID of the Union prompt for GENINPUT
CE99\$[ALL]	Employee and Record Type
PAUN_INDEX	FILE[x] element that PAUNx is open on
Y\$	Command returned from GENINPUT
GUI	“1” if in GUI mode
OSGUI\$	Standard GUI template

WORK_COMP:

This entry point will prompt for the Workers' Compensation code.

```
1805 WORK_COMP:
1810 ENTER ENTER FILE [ALL] , PAWC_INDEX , B8$ , CE99$ [ALL] , EG5$ , WORK_X0 , WORK_X1 ,
WORK_GUI , Y$ , GUI , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
PAWC_INDEX	FILE[x] element that PAWCx is open on
B8\$	Payroll information from JOBSx file
CE99\$[ALL]	Employee and Record Type
EG5\$	Employee default Workers' Compensation code
WORK_X0	X0 position of the Workers' Comp. Code prompt for GENINPUT
WORK_X1	X1 position of the Workers' Comp. Code prompt for GENINPUT
WORK_GUI	GUI ID of the Workers' Comp prompt for GENINPUT
Y\$	Command returned from GENINPUT
GUI	"1" if in GUI mode
OSGUI\$	Standard GUI template

UPDATE_JOB_TYPE:

This entry point will update the B5[ALL] cost types array with the correct amount

```
2105 UPDATE_JOB_TYPE:
2110 ENTER FILE[ALL],JOB_ID$,AMOUNT,B2[ALL],B5[ALL],SPREC
```

Variable	Description
FILE[ALL]	Standard file array
JOB_ID\$	Job/Phase/Cost Code entered by the user
AMOUNT	Extended amount of the line item
B2[ALL]	Array from JOBSx record containing overhead amount
B5[ALL]	Array from JOBSx record containing cost type amounts
SPREC	Precision for multiplication

PREPARE_CHECKS:

This entry point will update the various Payroll CJ special files with check information. We will pass in all variables at the current point of the call.

```
3005 PREPARE_CHECKS:
3010 ENTER
```

GETRATES:

This entry point returns the correct pay rate based on labor class and skill level.

```
5805 CHECK_ITEMS:
5810 ENTER FILE[ALL] , PAJC_INDEX, PACL_INDEX, PAEC_INDEX, RT1, CLASS$, CE6$, CE4$,
SK$, CE1[ALL] , RATE, EG2[ALL] , CL1$, CL2$, CL3$, CL4$, CL5[ALL] , CL6[ALL] , CL99$[ALL] ,
CL99[ALL] , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
PAJC_INDEX	FILE[x] index the PAJCx file is opened on
PACL_INDEX	FILE[x] index the PACLx file is opened on
PAEC_INDEX	FILE[x] index the PAECx file is opened on
RT1	Rate from PAJCx file
CLASS\$	Class entered by the user
CE6\$	Labor Code
CE4\$	Job/Phase ID
SK\$	Skill level entered by the user
CE1[ALL]	Store rate
RATE	Returns the rate calculated
EG2[ALL]	Default employee rate information
CL1\$	Class Code
CL2\$	Class Description
CL3\$	Skill Level IDs
CL4\$	Skill Level descriptions

Variable	Description
CL5[ALL]	Skill level pay rate
CL6[ALL]	Reserved for OSD
CL99\$[ALL]	Reserved for OSD
CL99[ALL]	Reserved for OSD
OSGUI\$	Standard GUI template

PACMNT.PUB

PACMNT.PUB is a public program that provides entry into GENCMNT.PUB for employee comment inquiry and maintenance.

```
ENTER FILE[ALL], EMPID$, X0$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
ITEMID\$	Employee ID for which comments are displayed and edited
X0\$	A = Maintenance mode; B = Inquiry only
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	PACMNT.PUB does not return any errors.

PAHCDRIL.PUB

PAHCDRIL.PUB is a public program that handles the drill down ability from General Ledger to Payroll check history information. This program will build a scratch file containing matching historical entries between the GLDRx and PAHCx files, then call PAHCCON.PUB to display the check history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	PAHCDRIL.PUB does not return any errors.

PAHDDRIL.PUB

PAHDDRIL.PUB is a public program that handles the drill down ability from General Ledger to Payroll check deduction history information. This program will build a scratch file containing matching historical entries between the GLDRx and PAHDx files, then call PAHDCON.PUB to display the check deduction history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	PAHDDRIL.PUB does not return any errors.

PAHEDRIL.PUB

PAHEDRIL.PUB is a public program that handles the drill down ability from General Ledger to Payroll check earning history information. This program will build a scratch file containing matching historical entries between the GLDRx and PAHEx files, then call PAHECON.PUB to display the check earning history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	PAHEDRIL.PUB does not return any errors.

PAHWDRIL.PUB

PAHWDRIL.PUB is a public program that handles the drill down ability from General Ledger to Payroll check withholding history information. This program will build a scratch file containing matching historical entries between the GLDRx and PAHWx files, then call PAHWCON.PUB to display the detail history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL], OSGUI$, GLJRSEQ$, YEAR_EXT$, MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	PAHWDRIL.PUB does not return any errors.

PAVOID.PUB

PAVOID.PUB is a public program that voids a payroll check in the PAHCxxx file. It backs the entries out of General Ledger and restores the employee pay data paid by the check. If Payroll is interfaced to Bank Reconciliation, the void is also recorded in the BRTRxxx file.

```
0050 ENTER MSG$,CHECK$,AMOUNT,RETURN$,CHECKAPP$,DUMMY,OSGUI$
```

Variable	Description
MSG\$	Returns a message string if a trapped error occurs
CHECK\$	Check number to void; if blank, user will be prompted
AMOUNT	Returns the amount of the check
RETURN\$	Employee ID associated with the check; if blank, user is prompted
CHECKAPP\$	Application ID (PA)
DUMMY	Not used
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	PAVOID.PUB does not return any errors.

PAVWDOC.PUB

PAVWDOC.PUB is a public program that provides entry into GENVWDOC.PUB for employee document inquiry.

```
ENTER FILE[ALL],EMPID$,X0$,OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
EMPID\$	Employee ID for which documents are displayed
X0\$	Not used
OSGUI\$	Template for GUI function

Exit Points	Explanation
EXIT	PAVWDOC.PUB does not return any errors.

PAYEARS.PUB

PAYEARS.PUB is a public program that builds a scratch file with active Payroll years running in OSAS. This interface is called from both the PASETUP.PUB function (F9 from the menu), which allows users to switch to prior year files, and from Change Fields, allowing users to change Payroll key fields across multiple years of Payroll files.

(Normal entry point)

This entry point is used to build an m-keyed file that contains a list of Payroll years. This is done by looking for the existence of the MASK\$ field in file names when searching thru the current dataset.

```
0100 ENTER F$,F0,MASK$,FILE[ALL]
```

Variable	Description
F\$	Name of m-keyed file built that contains PA years
F0	Channel F\$ is opened on
MASK\$	Mask field to use to search thru dataset (ex: "PAEGH.Y??", "PAEGH.Y*"). ? = Any single character * = Any number of characters
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	PAYEARS.PUB exits with no error.

CUSTOM_SCRATCH_FILE:

This entry point is used identically to how the normal entry point is used, with the exception that the scratch file built will contain 2 fields per record: the PA year and the word "NO ". This is used by the Change Fields function to build a list of years available for changing fields for.

```
0130 CUSTOM_SCRATCH_FILE:
```

```
0140 ENTER F$,F0,MASK$,FILE[ALL]
```

Variable	Description
F\$	Name of m-keyed file built that contains PA years
F0	Channel F\$ is opened on
MASK\$	Mask field to use to search thru dataset (ex: “PAEGH.Y??”, “PAEGH.Y*”). ? = Any single character * = Any number of characters
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	PAYEARS.PUB exits with no error.

POAUDIT.PUB

POAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Purchase Order module.

FILE_CHECK:

This entry point returns the total number of files in Purchase Order that has GL Accounts in them.

```
1010 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	POAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Purchase Order.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of POAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data file that contain GL accounts is the POORx file.

The tables that contain GL Accounts are: APGL, DM, DF, and MRGL.

POFOPUR.PUB

POFOPUR.PUB is a public program that will generate a BBj PDF purchase order form from properly formatted data files. This program will take the data and produce a purchase order looking similar to the OSAS laser form.

PRINT_FORM:

This entry point is used to print an invoice form after data is set up in temporary files.

```
0050 PRINT_FORM:
0060 ENTER PURCINFO$, POIFH$, POIFD$, FILE[ALL]
```

Variable	Description
PURCINFO\$	Template used for generic form information
POIFH\$	Template for purchase order header temporary file
POIFD\$	Template for purchase order detail temporary file
FILE[ALL]	Standard file array

Exit Points	Explanation
EXIT	POFOPUR.PUB normally exits with no error.

GET_TEMPLATES:

This entry point is to retrieve the template format that will be set up and used for general form information, as well as the temporary files built for use for printing purchase orders.

```
4010 GET_TEMPLATES;
4020 ENTER PURCINFO$, POIFH$, POIFD$
```

Variable	Description
PURCINFO\$	Template used for generic form information
POIFH\$	Template for purchase order header temporary file
POIFD\$	Template for purchase order detail temporary file

The following chart breaks down the PURCINFO\$ template.

Variable	Description
PRINTNAME:C(30*)	BBj Java name of printer to print to
FORMID:C(8*)	Form ID from scratch files to print
OUTPATH:C(65*)	Path and name of PDF file to generate if e-mailing or printing to file
LOGO:C(80*)	Path and name of logo image to print on form
COPYTEXT:C(15*)	Text to print if printing multiple copies of form
INDEXHEAD:N(2*)	FILE[x] element that header scratch file is opened on (ARIFHxxx)
INDEXDETL:N(2*)	FILE[x] element that detail scratch file is opened on (ARIFDxxx)
COMPADD1:C(30*)	Current company's Address Line 1
COMPADD2:C(30*)	Current company's Address Line 2
COMPADD3:C(30*)	Current company's Address Line 3
COMPADD4:C(30*)	Current company's Address Line 4
COMPADD5:C(30*)	Current company's Address Line 5
LOGOP:C(2*)	L=Left justified, other values = Centered
FORMPROG:C(12*)	"POFOPUR.PUB". Set in a variable so that the multi-copy form printing program knows what program to call to print invoices

See the Purchase Order file layouts for POIFHxxx and POIFDxxx for layouts of the temporary files and their templates.

POBLDQTY.PUB

POBLDQTY.PUB is a public program that creates a scratch file from PO Transactions files for use by the Inventory **Rebuild Item Quantities** function.

```
0060 ENTER FILE[ALL] , PASS$ , INQT$ , INSN$ , INVE$ , INLD$ , INLS$ , MSG$ , TASK , OPTION$
```

Variable	Description
FILE[ALL]	Standard file array used for the file channel number
PASS\$	Template array used for the data file channel number
INQT\$	INQTxxx file template (Quantity Totals)
INSN\$	INSNxxx file template (Serial Numbers)
INVE\$	INVExxx file template (Item Master)
INLD\$	INLDxxx file template (Location Detail)
INLS\$	INLSxxx file template (Transaction Lot/Serial Number)
MSG\$	Error messages returned to calling program
TASK	0 = Open files and prepare scratch file; 1 = Open files Only
OPTION\$	Application specific options and interfaces string

Exit Points	Explanation
EXIT	POBLDQTY.PUB does not return any errors.

The following chart breaks down the PASS\$ template.

Field	Description
INVE:N(2*)	Channel number on which the INVEx file is open
INLD:N(2*)	Channel number on which the INLDx file is open
QTSCR:N(2*)	Channel on which the INQTxxx scratch file is open (Quantity Totals file)
SNSCR:N(2*)	Channel on which the INSNxxx scratch file is open (Serial Number file)
ERRSCR:N(2*)	Channel on which the SCRT3.zzz error scratch file is open

POGENREQ.PUB

POGENREQ.PUB is a public program that writes PO requisitions to the POPQ_{xxx} file.

```
0050 ENTER ITEM$,DESC$,LOC$,QTY,UOM$,FACTOR,SOURCE$,GLACCT$
```

Variable	Description
ITEM\$	Item ID to requisition
DESC\$	Item description
LOC\$	Location ID to requisition
QTY	Quantity to requisition
UOM\$	Unit of measure to requisition
FACTOR	Conversion factor for the unit of measure
SOURCE\$	Application ID from which the requisition was generated
GLACCT\$	Inventory GL account number for the item

Exit Points	Explanation
EXIT	POGENREQ.PUB does not return an error.

POGENSOQ.PUB

POGENSOQ.PUB is a public program that prompts a user for requisition information. It is designed to be called from Sales Order. POGENSOQ.PUB differs from POGENREQ.PUB in that it prompts the user for information about the requisition being generated.

```
0050 ENTER ITEM$,DESC$,LOC$,QTY,UOM$,FACTOR,SOURCE$,IN,SER$,LOT,COST,OSGUI$
```

Variable	Description
ITEM\$	Item ID to requisition
DESC\$	Item description
LOC\$	Location ID to requisition
QTY	Quantity to requisition
UOM\$	Unit of measure to requisition
FACTOR	Conversion factor for the unit of measure
SOURCE\$	Application ID from which the requisition was generated
IN	IN interface flag: 0 - Sales Order is not interfaced to Inventory 1 - Sales Order is interfaced to Inventory
SER\$	Serial item flag: 1 - Nonserialized item 2 - Serialized item
LOT	Lotted item flag: 0 - not lotted 1 - lotted
COST	Unit cost of the item to be requisitioned
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	POGENSOQ.PUB does not return an error.

PORHDRIL.PUB

PORHDRIL.PUB is a public program that handles the drill down ability from General Ledger to Purchase Order receipt history information. This program will build a scratch file containing matching historical entries between the GLDRx and PORHx files, then call POIRHCON to display the receipt history entries that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL],OSGUI$,GLJRSEQ$,YEAR_EXT$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	PORHDRIL.PUB does not return any errors.

PORHTRAN.PUB

PORHTRAN.PUB is a public program that handles writing records to the Purchase Order receipt history file. This interface is called from routines that need to write new records to PORHx.

(Normal entry point)

This entry point is used when new records are to be written to the PORHx file. If PORH.SEQ\$ is set prior to calling PORHTRAN.PUB, the sequence number contained in that variable will always be used when writing a new record. If PORH.SEQ\$ is set to “”, a new sequence number will be generated and returned thru PORH.SEQ\$

```
0099 ENTER PORH,PORH$
```

Variable	Description
PORH	Channel PORHx file is opened on
PORH\$	PORHx string template (see the Purchase Order File Layouts)

Exit Points	Explanation
EXIT	PORHTRAN.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

RESERVE_PORH_SEQ:

This entry point is used when a sequence number is to be reserved so that it's not used until needed by the calling program. The sequence number is returned to the calling program thru the SEQ\$ variable. The next available number is then written back to the NXTSEQ record in PORHx.

```
3010 RESERVE_PORH_SEQ:  
3015 ENTER PORH,SEQ$,PORH$
```

Variable	Description
PORH	Channel PORHx file is opened on
SEQ\$	Next available sequence number in PORHx
PORH\$	PORHx string template (see the Inventory File Layouts)

RMACBFM.PUB

RMACBFM.PUB is a public program that handles **F6 Maintenance** for batches. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE[ALL], BARCH$, O2$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
BARCH\$	Batch ID entered by the user before pressing F6
O2\$	Application options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMACBFM.PUB does not return any errors.

RMACGCFM.PUB

RMACGCFM.PUB is a public program that handles **F6 Maintenance** for tax groups. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE [ALL] , CODE$ , O2$ , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Tax group entered by the user before pressing F6
O2\$	Calling application options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMACGCFM.PUB does not return any errors.

RMACTCFM.PUB

RMACTCFM.PUB is a public program that handles **F6 Maintenance** for tax classes. Use this program in entry programs to provide access to maintenance through the GENINPUT.PUB program.

```
ENTER FILE [ALL] , CODE$ , O2$ , OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
CODE\$	Tax class entered by the user before pressing F6
O2\$	Calling application options string
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMACTCFM.PUB does not return any errors.

RMAUDIT.PUB

RMAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Resource Manager module.

FILE_CHECK:

This entry point returns the total number of files in Resource Manager that has GL Accounts in them.

```
1010 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	RMAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILE$(ALL)
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILE\$(ALL)	2-dimensional array dimensioned to DATAFILE\$(1:APPFILES,1:2) DATAFILE\$(x,1) = Name of file to be opened DATAFILE\$(x,2) = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Resource Manager.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of RMAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: RMBAx, RMTDx, and RMTHx.

No tables in Resource Manager contains GL accounts.

RMCHKLY.PUB

RMCHKLY.PUB is a public program that checks for last-year GL files, and prompts the user for current- or last-year entry. It also updates the entry table in the CALLing application or removes the table, depending on the task requested.

```
0025 ENTER T$,TABLE$,TASK, LASTYEAR,EXT$,MESSAGE$,OSGUI$
```

Variable	Description
T\$	Table file name for the application
TABLE\$	Table ID
TASK	Task to be performed: 0 - create table and prompt for current- or last-year entry 1 - delete the table 2 - return LASTYEAR and the extension (no prompts)
LASTYEAR	Current- or last-year entry flag: 0 - current-year entry 1 - last-year entry
EXT\$	Extension for the GL files
MESSAGE\$	Returns a message string if an error occurs
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMCHKLY.PUB does not return an error.

RMCRTFIL.PUB

RMCRTFIL.PUB is a public program that will attempt to create a file with specifications found in an applications XXDATA.YYY file (where XX is the application ID and YY is the version), given the index to read and the file name to create. Encrypted files will also be created from this function.

```
0030 ENTER FILE[ALL] ,APP$,APPLDIR$,DESTFILE$,FILE$,VER$,MSG$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
APP\$	Application ID used to determine name of XXDATA.YYY file
APPLDIR\$	Not used
DESTFILE\$	Path and name of file to create
FILE\$	Key index to read from an XXDATA.YYY file
VER\$	Version of file used to determine the name of XXDATA.YY file
MSG\$	Error message if file could not be created

Exit Points	Explanation
EXIT	RMCRTFIL.PUB does not normally return any errors.
EXIT ERR	An error is returned when an error occurs.

RMGETVER.PUB

RMGETVER.PUB is a public program that returns the base version number from the xxCNVT file corresponding to the version supplied by the CALLing program. For example, to find the base version corresponding to version 4.53, CALL RMGETVER.PUB with VER\$ set to “4.53” and the program returns “4.50” as the base version number.

```
0030 ENTER APP$,VER$,EXT$
```

Variable	Description
APP\$	Application ID to check
VER\$	Version number to convert
EXT\$	Returns the base version number

Exit Points	Explanation
EXIT	RMCHKLY.PUB normally exits without an error.
EXIT ERR	An error is returned when an error occurs.

RMMERGE.PUB

RMMERGE.PUB is a public program that merges two data files together. It is generally used during installation to merge a file update into a base data file.

```
0090 ENTER SFILE$,DFILE$,SUPPRESS_ERROR,NO_ERASE_SOURCE,OSGUI$
```

Variable	Description
SFILE\$	File to be merged
DFILE\$	File into which the source file is merged
SUPPRESS_ERROR	If set to >0, error message will not print on screen if file doesn't exist
NO_ERASE_SOURCE	If set to >0, source file will not be erased after merging is complete
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMMERGE.PUB does not return an error.

RMMNUMRG.PUB

RMMNUMRG.PUB is a public program that will attempt to merge a list of entries from an application's XXMNU.MRG text file into the XXMN file. A variety of options and parameters can be set in the merge file to accomplish a variety of menu configuration. This process was implemented for enhancement applications that change an application's menu structure, to avoid having a copy of the entire menu file in the enhancement application.

```
1050 ENTER FILE[ALL], MENUFILE$, OSGUI$
```

Variable	Description
FILE[ALL]	Standard file array
MENUFILE\$	Menu merge template
OSGUI\$	Standard GUI template

Exit Points	Explanation
EXIT	RMMNUMRG.PUB normally exits with no error.

The following chart breaks down the MENUFILE\$ template and its possible options:

Field	Description
APPL:C(2*)	Application ID's menu to merge data into
MENU:C(6*)	Menu ID to merge into
ACTION:C(2*)	Action to perform: "AL" = Append a line to end of menu record "DL" = Delete a line given a program name and parameter "IA" = Insert line after given a program name and parameter "IB" = Insert line before given a program name and parameter "RL" = Replace line given a program name and parameter
TARGETLINE:C(1*)	Not currently used
TARGETPROG:C(1*)	Program name to look for
TARGETPARAM:C(1*)	Menu's parameter to look for
DESC:C(40*)	Description to add to menu record

Field	Description
TYPE:C(1*)	Program type to add to menu record
PROG:C(12*)	Program name to add to menu record
PARAM:C(2*)	Parameter to add to menu record
SDESC:C(20*)	Short description to add to menu record
PATH:C(60*)	Path (if necessary) to add to menu record - Used for documents only

RMPRTXT.PUB

RMPRTXT.PUB is a public program designed to parse a text file that takes the format of a Microsoft .INI file format. Calling programs can retrieve section headers only, a specific section of the INI file, or the entire file. Up to three separate sections can be retrieved at a given time.

```
0060 ENTER TEMP1$,TEMP2$,TEMP3$,FILENAME$,ERRMSG$
```

Variable	Description
TEMP1\$	“GET SECTIONS” = Get all section names and return in TEMP1\$, separated by line feeds “” = Get all fields in file, not separated by section, and return in template form thru TEMP1\$ “xxxxx” = Finds section named xxxxx and returns field names and data in template format
TEMP2\$	“xxxxx” = Finds section named xxxxx and returns field names and data in template format
TEMP3\$	“xxxxx” = Finds section named xxxxx and returns field names and data in template format
FILENAME\$	Text file to open and read data from
ERRMSG\$	Error message returned to calling program

Exit Points	Explanation
EXIT	RMPRTXT.PUB does not return any errors.

An .INI file is a standard text file that holds information for a given function. It can be divided into multiple sections, which each section having its own fields. An example of an INI file is as follows:

```
[System Paths]
Install Directory=d:/
Base Directory=../

[Prefix Paths]
Data 1=../DATA/
Data 2=
Data 3=
Sample=../sample/
Graphics=../gui/
System=../sysfil/
Report Writer=../RWdata/
Utilities=C:/OSAS/75P2BBJ/bbj/utlils/std/

[Other Paths]
CplLstDir=C:/OSAS/752SHIP/bbj/bin/
RebuildDir=C:/OSAS/752SHIP/bbj/utlils/mkrecover/
Documents=C:/OSAS/752SHIP/Document/
DocumentShare=C:/OSAS/752SHIP/Document/

[CD Key]
Key=ABCDE-0123456789ABCDEF012
```

In this example, the file has 4 different sections, which are noted by the text inside of the '[]'. Under each section are field names and data. Each field name can only occur once within a section, and typically, only appears once in the entire file.

Example 1: If TEMP1\$ contains "Prefix Paths", TEMP1\$ would return the following template: "DATA1:C(9*),DATA2:C(1*),DATA3:C(1*),SAMPLE:C(10*),GRAPHICS:C(7*),SYSTEM:C(10*),REPORTWRITER:C(10*),UTILITIES:C(30*)". Each variable of TEMP1\$ can then be access to reflect the data in the file.

RMUPCHGF.PUB

RMUPCHGF.PUB is a public program that merges data for the **Change Fields** function for an application (OSFD.xx and OSFH.xx) into OSFD and OSFH. It is generally used during application installation only by RMAPPMRG.PUB after an application is installed.

```
0090 ENTER APP$,OSGUI$
```

Variable	Description
APP\$	Application ID
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMUPCHGF.PUB does not return an error.

RMUPDICT.PUB

RMUPDICT.PUB is a public program that merges data dictionary information for an application into the BASIS data dictionary. It is generally used during application installation only by RMAPPMRG.PUB after an application is installed.

```
0090 ENTER APP$,OSGUI$
```

Variable	Description
APP\$	Application ID
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMUPDICT.PUB does not return an error.

RMUPGEN.PUB

RMUPGEN.PUB is a public program that merges GENERAL Report Writer data dictionary information for an application (GEN6MST.xx_D and GEN6MST.xx_R) into GEN6MST. It is generally used during application installation only by RMAPPMRG.PUB after an application is installed.

```
0090 ENTER APP$,OSGUI$
```

Variable	Description
APP\$	Application ID
OSGUI\$	Template for GUI functions

Exit Points	Explanation
EXIT	RMUPGEN.PUB does not return an error.

SOAUDIT.PUB

SOAUDIT.PUB is a public program that handles a variety of functions for the GL Account Audit Report. This program is called by RMPRAUD and RMPRAUD1, and handles all GL accounts found in the Sales Order module.

FILE_CHECK:

This entry point returns the total number of files in Sales Order that has GL Accounts in them.

```
1010 FILE_CHECK:  
1030 ENTER APPFILES
```

Variable	Description
APPFILES	Number of files that contains GL Accounts

Exit Points	Explanation
EXIT	SOAUDIT.PUB normally exits with no error.

FILE_DATA:

This entry point will set the exact file names into an array and return those names to the calling program, which will in turn open the returned files.

```
1510 FILE_DATA:  
1530 ENTER APPFILES,DATAFILES$[ALL]
```

Variable	Description
APPFILES	Number of files that contains GL Accounts (retrieved from using the FILE_CHECK entry point)
DATAFILES\$[ALL]	2-dimensional array dimensioned to DATAFILES\$[1:APPFILES,1:2] DATAFILES\$[x,1] = Name of file to be opened DATAFILES\$[x,2] = "1" if file is encrypted, "" if not

AUDIT:

This entry point will perform the auditing of data files within Sales Order.

```
2010 AUDIT:
2030 ENTER GLMA,MSG$,OUTPUT$,FILE[ALL],TEMPLATE$[ALL]
```

Variable	Description
GLMA	FILE[GLMA] element the GLMAx file is opened on
MSG\$	Returns an error message if audit was able to perform
OUTPUT\$	Template holding print information for report
FILE[ALL]	Standard file array
TEMPLATE\$[ALL]	Array holding each string template of the corresponding files returned from the FILE_DATA entry point. Each template is on an index that corresponds to the FILE[ALL] array where the file is opened on

The OUTPUT\$ template has the following format:

Field	Description
PCHAN:N(3*)	Output channel for report (Set to variable F)
DATE:C(10*)	Current workstation date
TIME:C(8*)	Current OS time
CNAME:C(40*)	Company name
TITLE:C(40*)	Title of report (Set to variable OT\$)
PAGE:N(4*)	Current page number of report
APPID:C(2*)	Current application ID
APPNAME:C(40*)	Current application name
LNCOUNT:N(3*)	Current line count
PGLINES:N(3*)	Number of lines per page
FIRSTPG:N(1*)	Set to 1 when on page 1, otherwise set to 0
FILESEC:N(1*)	Set to 1 when printing a file section of the report
TABLESEC:N(1*)	Set to 1 when printing a table section of the report

The AUDIT entry point of SOAUDIT.PUB performs the following functions:

- Retrieves templates to use by GL Audit report by calling RMPRAUD.PUB::DIM_VARS
- Determines if GLMAx exists to validate GL accounts from. If the file does not exist, only blank GL accounts are reported back to the audit report
- Retrieves application options to make sure application is interfaced to GL
- Finds the FILE[ALL] elements that each application data file is opened on
- Audits GL Accounts of each data file, checking if it exists in the GLMAx file, as well as the accounts status
- Audits GL Accounts in each table that can contain GL Accounts, checking if it exists in the GLMAx file, as well as the accounts status

The data files that contain GL accounts are: SOBLx, SORLx, and SOTBx.

The table that contain GL Accounts is the ARGLx table.

SOBLDQTY.PUB

SOBLDQTY.PUB is a public program that creates a scratch file from Sales Order Transactions files for use by the Inventory **Rebuild Item Quantities** function.

```
0060 ENTER FILE[ALL],PASS$,INQT$,INSN$,INVE$,INLD$,INLS$,MSG$,TASK,OPTION$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
PASS\$	Template array used for the data file channel number
INQT\$	INQTxxx file template (Quantity Totals file)
INSN\$	INSNxxx file template (Serial Numbers file)
INVE\$	INVExxx file template (Item Master file)
INLD\$	INLDxxx file template (Location Detail file)
INLS\$	INLSxxx file template (Transaction Lot/Serial Number file)
MSG\$	Error messages returned to calling program
TASK	0 = Open Files and prepare scratch file; 1 = Open Files Only
OPTION\$	Application specific options and interfaces string

Exit Points	Explanation
EXIT	SOBLDQTY.PUB does not return any errors.

The following chart breaks down the PASS\$ template.

Field	Description
INVE:N(2*)	Channel number on which the INVEx file is open
INLD:N(2*)	Channel number on which the INLDx file is open
QTSCR:N(2*)	Channel on which the INQTxxx scratch file is open (Quantity Totals file)
SNSCR:N(2*)	Channel on which the INSNxxx scratch file is open (Serial Number file)
ERRSCR:N(2*)	Channel on which the SCRT3.zzz error scratch file is open

SOCRLCK1.PUB

SOCRLCK1.PUB is a public program that checks the customer balance against the credit limit. It includes current transactions and cash receipts in the balance.

```
100 ENTER CUST$,CREDLIM,SOTHFNUM,ARCRFNUM,FILE[ALL],CURRTRAN$
```

Variable	Description
CUST\$	Customer ID
CREDLIM	Customer credit limit
SOTHFNUM	FILE[] array index for the SOTHxxx file
ARCRFNUM	FILE[] array index for the ARCRxxx file
FILE[ALL]	Standard file array used for the data file channel number
CURRTRAN\$	Current order number

Exit Points	Explanation
EXIT	SOCRLCK1.PUB normally exits with no error.
EXIT ERR	An error is returned when an error occurs.

SOKHDRIL.PUB

SOKHDRIL.PUB is a public program that handles the drill down ability from General Ledger to Sales Order kit history information. This program will build a scratch file containing matching historical entries between the GLDRx and SOKHx files, then call SOKTHCON:: GLDRILL to display the kit history detail record that made up the entry to GLJRx.

```
0060 ENTER FILE[ALL], OSGUI$, GLJRSEQ$, YEAR_EXT$, MSG$
```

Variable	Description
FILE[ALL]	Standard file array
OSGUI\$	Standard GUI template
GLJRSEQ\$	Sequence number of GLJRx file being drilled on
YEAR_EXT\$	Year of GLJRx file user is viewing.
MSG\$	Set if no history records were found

Exit Points	Explanation
EXIT	SOKHDRIL.PUB does not return any errors.

SOPRIOP.PUB

SOPRIOP.PUB OPENS files and DIMensions string templates used in invoice printing.

```
0025 ENTER FILE[ALL] , HEAD$, DATA$, O2$, T$, PLP, ONLINE, TRAN_LEN, HEAD, KITON, STAT$
```

Variable	Description
FILE[ALL]	Standard file array used for the file channel numbers
HEAD\$	String template for invoice header printing
DATA\$	String template for invoice line-item printing
O2\$	Sales Order options string
T\$	Application tables file name
PLP	Invoice form type flag: 0 - preprinted forms 1 - plain paper 2 - 8½-by-11-inch forms
ONLINE	Online invoice flag: 0 - batch invoice printing, 1 - online invoice
TRAN_LEN	Length of the transaction number (set to 8)
HEAD	FILE[] array element for the SOTHxxx file
KITON	Kit detail flag: 0 - do not print kit detail on the invoice 1 - print kit detail on the invoice
STAT\$	Valid statuses to print (usually set to “3458” in SO): 1 - new order 2 - picked order 3 - verified order 4 - returned order 5 - invoiced order 6 - backordered order 8 - quote

Exit Points	Explanation
EXIT	SOPRIOP.PUB does not return an error.

SOTRAN.PUB

SOTRAN.PUB is a public program that will generate, validate, and reserve a transaction number for the SOTHx file. The returned number should be used as the transaction number before writing a new record to the SOTHx file.

```
0100 ENTER FILE[ALL] , TRAN, HEAD, CT1, CT2, ORDER_NO, R0$, SYST_GEN, Y$
```

Variable	Description
FILE[ALL]	Standard file array used for the data file channel number
TRAN	FILE[TRAN] index that the SOTDx file is opened on
CT1	FILE[CT1] index that the ARCTx file is opened on
CT2	FILE[CT2] index that the ARCTx file is opened on. This must be different than CT1
ORDER_NO	Order number passed in if not using system generated order numbers
R0\$	Batch/Transaction number from SOTHx. Should contain batch ID set by the calling program + “ ” or “00000000”. The transaction is returned in the last 8 characters.
SYST_GEN	System generated flag from Sales Order Options (O2\$(6,1))
Y\$	Set from GENLOCK.PUB in case of error conditions

Exit Points	Explanation
EXIT	SOTRAN.PUB does not return any errors.

CHAPTER 4

4

OSAS 4.5x Upgrade Procedure	4-3
Converting to 6.xx Graphical	4-19
Converting to 7.0x	4-59

Upgrading OSAS

This chapter covers the procedure to upgrade or convert an application from version 4.5x to version 6.xx., to upgrade or convert an existing character-based application to run in both character and graphical modes, or to upgrade or convert an existing version 6.x application to run on both PRO/5 and BBj language environments.

OSAS 4.5x Upgrade Procedure

Follow the steps on this checklist to upgrade or convert an application from version 4.5x to version 6.xx. Standards from version 4.5x apply unless stated otherwise.

Application _____

Done	Task	Initials/Date
	Convert programs to PRO/5	
	Merge new GENERROR and GENLOCK routine	
	Merge new GENPC, GENPC1 and GENPC2 routine	
	Merge new GENINPUT routine	
	Merge and implement new GENOPEN routine	
	Merge and implement new GENLINE routine	
	Convert xxWIND file	
	Implement user-defined numeric masking	
	Implement error 43 protection	
	Change Global Inquiry window definitions	
	Convert xxHELP file	
	Convert application Tables file	
	Implement 3-character company IDs	
	Implement 3-character state IDs	
	Implement 20-character phone numbers	
	Change "Record Not Found" messages	
	Add Tables List	
	Create Options and Interfaces file	
	Update sample data	

Done	Task	Initials/Date
	Change General Ledger interface	
	Rewrite conversion programs	
	Implement new report standards	
	Create Application Message (xxMS) file	
	Create GENERAL Report Writer files	
	Create BASIS Data Dictionary files	
	Create Change Field files	
	Create Template Merge files	
	Add user-defined fields for master files	
	Add documents for master files	
	Change xxLAST.TXT file	
	Complete the graphical conversion	

Convert Programs to PRO/5

All programs from previous versions must be converted to PRO/5. Follow these steps to convert your programs:

1. If the programs you want to convert are at a BBx Progression/4 level, you will not need to run any conversion programs on them to bring them to a PRO/5 level. These programs will run under PRO/5. The first time you SAVE a program under PRO/5, it will be automatically converted to a PRO/5 level, and you will not be able to load that program under BBx Progression/4.
2. Copy the contents of the utils directory on the RM media to a utils directory.
3. Add the utils directory to the prefix, using the Directories function in RM.
4. Change to the directory that contains the programs to be converted.
5. CALL “_p4conv” to convert your programs. The programs are converted in place and overwrite the P/3 versions.

Merge New GENERROR and GENLOCK Routine

Merge the new GENERROR and GENLOCK routines into your programs. The GENERROR routine has been changed to restore precision on a RESET and to handle the optional message file (see the General Routines section for more information). Both routines pass OSGUI\$ so that messages can appear in graphical mode.

Merge New GENPC, GENPC1 and GENPC2 Routines

Merge the new GENPC routines into your programs. Follow these steps to merge the routine and change your programs:

1. Change the GENPC signature line in your programs, adjusting the line numbers to be removed to include the GENPC1 and/or GENPC2 routines.
2. RUN or CALL the _update utility to merge the new routine from GENLIB (which is on the Software Development Utilities diskette).
3. Change the logic for the GOSUBs to GENPC. You no longer need to
 - CALL GENPC to load the O9\$ variable, unless you need it locally
 - set F\$, F1 or F2 for the tables file before calling GENPC
 - OPEN the tables file before calling GENPC

Merge New GENINPUT Routine

Programs that use GENINPUT must be checked for these things:

1. GL Period fields should use X3=10.
2. Year fields should use X3=11.
3. Period/Year fields should use X3=12.
4. GL Account Number fields should use X3=13.

Using these parameters in GENINPUT decreases the size of the program you are updating. Remove validity tests that check whether a period or year is valid, and CALLs to GENERERROR if a value is invalid. You must still verify a GL account against the GLMAxxx file, however. The GENINPUT CALL only assures that the user uses the correct format for the account number, not that the account exists.

You must also include these GUI components to all GENINPUT assignment lines:

- Add GUI_ID for all GUI control IDs
- Add GUI_RADIO\$ for all input fields where a radio group is used

In programs where a control bar is found on the character screen, such as in HLE programs or inquiry programs that allow for cycling through records in a master file, define the two-dimensional array GUI_BUTTON\$[ALL] as follows:

- GUI_BUTTON\$[X,1] contains the GUI control ID of the buttons as defined in the resource file
- GUI_BUTTON\$[X,2] contains the keystroke letter that the user can press to operate the control

When the user chooses graphical mode, all command bar screen commands must be activated by all three of these methods:

- clicking on the button with the mouse
- pressing the ALT key and the underlined letter
- pressing the underlined key on the keyboard (without the ALT key)

Merge GENOPEN Routine

Merge the GENOPEN routine into your programs. This routine opens all data files in a consistent manner, and supports other functions, such as retrieving a template for a given file, opening a file on the first available channels, searching for a specific file within the FILE[ALL] array, and closing a file.

Merge GENLINE Routine

Merge the GENLINE routine into your programs. It CALLs GENGULIN.PUB if OSAS is in graphical mode, and it also passes FLD_HLP\$ and PGM(-2) into both GENLINE.PUB and GENGULIN.PUB. FLD_HLP\$ now works in scroll regions.

Convert xxWIND File

All xxWIND files must be converted before they will work with Resource Manager versions 5.00 and above. To convert these files, follow these steps:

1. Copy 50WIND from the RM media utils directory to the progRM directory.
2. RUN "50WIND".
3. Enter the ID of the application you want to convert.
4. Repeat step 3 for each application needing conversion.
5. Exit 50WIND.
6. Copy new xxWIND file(s) onto your media.

Old xxWIND files have the extension .OLD. The new windows definitions have the ability to return the value from any of the displayed columns to GENINPUT. In previous versions you could return only the contents of column one. The program 50WIND is also available under the name SD50WIND on the version 6.0 Software Development Utilities diskette.

Implement User-Defined Numeric Masking

Programs that deal with numeric values should be converted to use the numeric masking defined by the user in the Resource Manager **Company Information** function.

Users can define the precision of several numeric types: dollar amounts, quantities, unit prices, unit costs, hours, rates and credit limits. These numeric fields have a 14-character mask, which Resource Manager builds for you. The precisions and masks are stored in global string variables as follows:

Number Type	Mask	Precision
Regular dollar amounts	" +DOLL "	" -DOLL "*"
Quantities	" +IQTY "	" -IQTY "
Unit prices	" +INUP "	" -INUP "
Unit costs	" +INUC "	" -INUC "
Hours	" +HOUR "	" -HOUR "
Rates	" +RATE "	" -RATE "
Credit Limits	" +CRED "	" -CRED "
GL dollar amounts	" +GDOL "	" -GDOL "*"
GL dollar amounts with commas	" +GCOM "	" -GCOM "*"

*Regular dollars and GL dollars always have the same precision.

Note: Some numeric fields can be of hard-coded precision according to the needs of the program (for example, variance percentages) and are not defined by the user.

Follow these steps to implement the preferred method for handling these variable precisions:

1. Determine the precisions required by the program based on the numeric types used in the program and assign them to variables. For example, in a program that includes dollar amounts and quantities

```
LET DPREC=NUM(STBL (" -DOLL" ))
LET QPREC=NUM(STBL (" -IQTY" ))
```

2. Determine the maximum precision required by these numerics using

```
LET MPREC=MAX(DPREC, QPREC)
```

3. Because PRECISION is reset when a program is RUN, set the maximum precision at the beginning of each program in the chain:

```
PRECISION MPREC
```


4. Use the ROUND() function when performing multiplication or division tasks, including any BB^x functions that involve multiplication or division, such as INT(), FPT() or MOD():

```
EXTPRICE=ROUND (UNITPRICE*QUANTITY,DPREC)
```

Implement Error 43 Protection

You must merge the GENNUM routine into programs that display or print numerics. This general routine uses line 10000–10009, and handles numeric masking, using a function called FNNUM\$(). This function accepts a numeric value and a mask, and returns a string containing the masked numeric value or asterisks (*) if the value overflows the mask. To use this function within a program, follow these steps:

1. Assign the masks used in the program to string variables. For example,

```
DOLLMASK$=" "-STBL (" +DOLL" )
PCTMASK$=" -####.00"
```

The global strings for user-defined masks do not contain minus signs (-). You must add them, in keeping with OSAS standards, to the beginning of the mask for screen display or to the end of the mask for report output.

2. When displaying or printing a numeric value, use the FNNUM\$() function:

```
PRINT @ (12,4) , FNNUM$ (QUANTITY, QTYMASK$) ,
```

or

```
PRINT (F) @ (45) , FNNUM$ (PERCENT, PCTMASK$) ,
```

Change Global Inquiry Window Definitions

Use the **Global Inquiry Definitions** function on the Resource Manager **System File Maintenance** menu to update the numeric fields in Global Inquiry window definitions to use a user-defined numeric mask. You must update definitions that contain formulas or other fields that are not defined in the Report Writer data dictionary. You must also change all definitions that use Report Writer information, as those files are not shipped starting with version 6.1x.

Convert xxHELP File

All xxHELP files must be converted before they will work with Resource Manager 6.1x and later, if those help files were used on version prior to version 5.0. To convert these files, follow these steps:

1. Copy HELPCNVT from the utils directory on the RM media into the progRM directory.
2. RUN "HELPCNVT".
3. Enter the ID of the application you want to convert.
4. Repeat step 3 for each application needing conversion.
5. Exit HELPCNVT.
6. Copy new xxHELP file(s) onto your media.

Old xxHELP files have the extension .OLD. The new help screens can contain more than five lines per field.

Convert Application Tables File

All xxTB files must be converted to allow for an 8-character table ID if the tables were used on systems prior to version 5.0x. The 8-character table ID is the primary key of the file, and is necessary to allow for the 3-character company ID in company-specific tables. In addition, logic in your programs must also be adjusted to specify the 3-character ID when reading a company-specific table, or to use 3 spaces instead of 1 when reading nonspecific tables.

Implement 3-Character Company IDs

If the company ID is included in a field in any file—such as the GL Master file—the logic must be changed to increase the ID to 3 characters.

Implement 3-Character State IDs

If the state ID is included in a field in any file—such as the AR Customer file—the logic must be changed to increase the ID to 3 characters. This ID should also be verified against the OSST file.

Implement 20-Character Phone Numbers

If a phone number is included in a field in any file—such as the AR Customer file—the logic must be changed to increase the ID to 20 characters. The mask should come from the Country ID associated with the current record. If there is no Country ID, then the mask stored in STBL(“PHONEMASK”) should be used.

Change “Record Not Found” Messages

Change messages using language similar to *Record Not Found* or *xxxxx Not on File* to use the O7=8; GOSUB GENERROR logic. The message appears consistently if X\$ contains the value entered by the user and X4 contains the field length.

Add Tables List

Add the **Tables List** (Resource Manager program RMPRTB) to the **Master File Lists** menu for the application.

Create Options and Interfaces File

Application options and interfaces are no longer handled through an xxACOP.PUB program. Instead, you must build an application Options and Interfaces (xxOI) file.

An xxOI file is created and maintained through the **Options and Interfaces** function in Resource Manager. An option to allow editing of the xxOI file appears on the command bar when the Software Development Utilities are installed on your system. Refer to the *Software Development Utilities User's Manual* for a description of the prompts for editing the options and interfaces.

Update Sample Data

Write a routine to change the sample data dates to 2003. This is the new date standard for version 6.5x.

Change General Ledger Interface

Change programs that create General Ledger Journal entries to include the transaction date and the posting date.

Rewrite Conversion Programs

Most base applications have conversion programs up to version 6.5x. Therefore, you may need to also add a file conversion for your application.

If you have not re-written conversion programs to the new standard of chained conversion programs, follow these steps to use the new conversion structure (see “Data File Conversion” on page 2-125 for a more detailed discussion):

1. Change your existing conversion programs. (The skeleton program SDSKCNVT on the Software Development Utilities diskette can be used as a model for the new program format.)
2. Create the version 4.5x to 6.5x conversion programs, using SDSKCNVT.
3. Add the conversion data file records for each version (since 3.2x) of your application to the OSCNVT file, using the SDACCNVT program. This information is used to verify that the necessary source files are available at each conversion level.
4. Write a small program to separate the records for your application into an xxCNVT file.
5. Copy the xxCNVT file to the application program directory on the media.
6. Include an xxLAST task to merge the xxCNVT file into OSCNVT during installation.
7. Create the xxCNVT.HDR file using the Software Development Utilities.

Implement New Report Standards

Make the following changes to reports, lists and logs print in your applications:

1. Locate the final **FF** mnemonic in the report and add a comma to the end of the print line to suppress the line feed after the final form feed. This helps in report alignment and in output to laser and other page-fed printers.
2. Banner page printing is automatic if the user selects the option to print them in Resource Manager. When forms are printed, however, banner pages should be suppressed to avoid wasting forms. To suppress banner pages, use this example:

```
LET DUMMY$=STBL ("NO_BANNER" , "ON")
```

3. Modify report programs to print reports in compliance with the Screen and Report Standards documented later in this chapter.

Create Application Message (xxMS) File

You can choose to store application error and field help messages in an application message (xxMS) file, instead of hard-coding them into your programs. Separating error message text from the program logic provides for standard messages; allows easier modification of messages; and takes the first of many steps toward interface-independence, which eases language translations and is a requirement of event-driven user interfaces.

To implement an application message file, follow these steps:

1. Use the System Messages function in Resource Manager to add the system message to the xxMS file.
2. Assign the MX8 or MX9 variables to the message number you want to display, and GOSUB GENERROR.

The MX8 variable carries a message number to replace the X8\$ variable, and the MX9 variable replaces the X9\$ variable. You may still use X8\$ and X9\$ and these methods can coexist, although you should choose one method or the other.

Create GENERAL Report Writer Files

The GENERAL Report Writer Dictionary consists of file definitions and reports for use with GENERAL Report Writer. These files are created using functions within GENERAL Report Writer, and are then split up by running GNSPLIT. This program is found on the GENERAL Report Writer media, and splits an application's dictionaries and reports into GEN4MST.xx_D and GEN4MST.xx_R files. These files can then be included in the xxRWDATA.ZIP files for installation. The installation process then merges these files into GEN4MST.

Create BASIS Data Dictionary Files

The BASIS Data Dictionary is a set of data files that describe the contents of the data files used in OSAS for use with the ODBC driver that is included in the OSAS ODBC Kit. The dictionaries are also used by the Software Development Utilities to print file layouts, create string templates, and so on.

Use the functions in the Software Developers Kit to set up and maintain these data dictionaries. Include these files in the xxRWDATA.ZIP file for installation. The installation process will merge these files into the full OSAS data dictionary.

Create Change Field Files

The **Change Fields** functions in each application allows the user to change a master code in all applications, such as a customer ID or vendor ID. The information is pulled from the OSFD and OSFH files about which fields are in what files, and what their relative field position is within each record.

If your application contains new codes that you want the user to be able to change, or if your application files contain master codes from existing OSAS applications, use the **Change Fields** function in Resource Manager to enter those codes. Use the Software Development Utilities **Create Change Fields File** function to split your changes into the OSFD.xx, and OSFH.xx (where xx is your application ID). Then include these files in the xxSYSFIL.ZIP file for installation. The installation process merges these files into OSFD and OSFH.

Create Template Merge Files

You must create string templates for all data files. To do so, use the OSAS Software Development Utilities or ODBC Kit to set up and maintain the BASIS data dictionaries for each file, and then use the Software Development Utilities to create the template information file from the data dictionaries.

When you use the Software Development Utilities application to set up field information, you must enter a template name. This name is used by the **Create/Update Template File** function to create an application-specific xx.TPL file. You can then include this file in the xxSYSFIL ZIP file on the distribution media. When the user installs the application, the installation program merges the file into OSTPL.

Add User-Defined Fields for Master Files

Many master files have a master user-defined field file associated with them, which allows users to set up and store additional information for each master file record. For example, customers, vendors, and employees all have user-defined field files associated with them.

To add user-defined fields to your application, follow these steps:

1. Create a file using the name of the master file with a .UF extension (for example, ARCUx.UF and APVEx.UF). Make sure the key structure follows the structure of existing user-defined files. The first field of the file is the primary key of the master file, and the second field is always a length of 15.
2. Create the necessary string template for the .UF file, and make sure to include this template in the application xx.TPL merge file.
3. Add logic to the master file's maintenance programs to call the generic GENUSRFM.PUB program.

Add Documents for Master Files

Many master files have a master documents file associated with them, which allows users to associate documents with individual records in the master file. For example, customers, vendors, and employees all have document files associated with them.

To add user documents to your application, follow these steps:

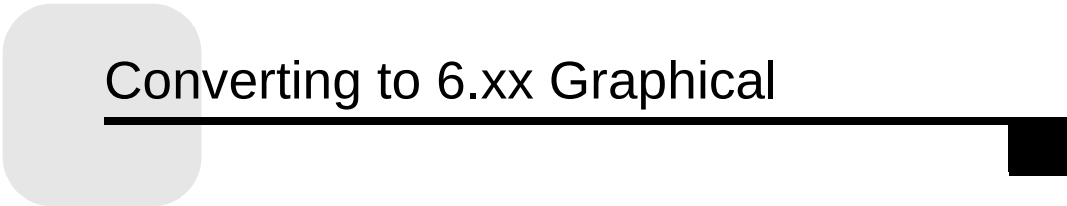
1. Create a file using the name of the master file with a .UD extension (for example, ARCUx.UD and APVEx.UD). Make sure the key structure follows the structure of existing user document files. The first field of the file is the primary key of the master file, and the second field is always a length of 3.
2. Create the necessary string template for the .UD file, and make sure to include this template in the application xx.TPL merge file.
3. Add logic to the master file's maintenance programs to call the generic GENACDOC.PUB program.
4. If you plan to use GENVWDOC.PUB, you must create an application specific xxVWDOC.PUB program to prepare for the call to GENVWDOC.PUB. This program is most commonly called by the **Shift+F2** Information functions. Use one of the existing application's xxVWDOC.PUB programs as a model when creating a program for a new application.

Change xxLAST.TXT File

You no longer need to have an xxLAST.TXT task to merge application-specific information into these standard OSAS files: OSCNVT, OSCNVT.HDR, OSFRM, OSTPL, OSORDER, GENERAL Report Writer Data Dictionaries, and BASIS Data Dictionaries. You also do not need to use the xxLAST.TXT file to merge the new Change Field information into OSFD and OSFH. The installation process will automatically look for these application-specific files and, if they exist, merge them into the necessary OSAS files.

Complete the Graphical Conversion

If you want your application to support both character and graphical modes, you must convert all of your application programs to support both modes. See “Converting to 6.xx Graphical” on page 4-19 for details on how to convert your application to support both character and graphical modes.



Converting to 6.xx Graphical

Use the following information to upgrade or convert an existing character-based application to run in both character and graphical modes.

Graphical Terminology

Before you start converting your application to run in both character and graphical modes, you must understand certain Windows terminology. Below are some common terms.

Form

Any window that is displayed, including the equivalent of character-based screens, error message boxes and other dialogue boxes are known as *forms*. Open Systems uses the terms *screen*, *window*, and *form* interchangeably.

Control

Every feature on a form—including the pull-down menus, tool buttons, status bar, list buttons, radio buttons, input fields, grids, and so on—are known as *controls*. Each control carries an ID that is unique to the form on which it resides.

Focus

Unlike in character mode where you must move one field at a time to get from field to field, when you are in graphical mode, several fields are available to you at one time. A user may click on any field to jump to it at any time (when the field is enabled). The act of moving the cursor to a control in response to a mouse click is known as *getting focus* or *gaining focus*. Conversely the act of leaving a control in response to a mouse click elsewhere is known as *losing focus*.

When a form first appears, the first control gets focus, and an edit cursor appears or the field is highlighted in some manner. As you leave that control to go to another, the first control loses focus and the new control gains focus.

Events

The process of gaining and losing focus is just one of many *events* trapped by Visual PRO/5. Other events include clicking on a check box, selecting an item from a list button, or clicking on a button.

You need not worry about how to trap events. GENINPUT.PUB has a sister program called GENGUINP.PUB that handles most events that can happen within a standard OSAS screen. It then interprets these events and passes the information back in the X\$, Y\$, and X variables among others, which your program can use much as it would with the character-based GENINPUT.PUB program.

CONTEXT

Visual PRO/5 has a method for tracking multiple graphical screens that are open at the same time. It assigns each graphical form its own CONTEXT number. You can use the CONTEXT mnemonic to assign the context before you open a new form, or to move between open forms. This technique is similar to that of the GOTO mnemonic used to jump between multiple character windows (although you cannot assign a value to a character screen before you open it).

Refer to the Visual PRO/5 manual for more information on CONTEXT.

How OSAS Does GUI

Windows-based applications written in languages like Visual Basic® use *object oriented programming*. In this programming method, the objects or controls placed on a form (as well as the form itself) contain code that is executed when events involving the control occur (such as gaining or losing focus).

OSAS code is written in a top-down structure, meaning that the code is essentially executed from beginning to end, and is not tied to specific fields or events. Each of these coding structures has advantages and disadvantages.

The goal of Open Systems in developing the graphical interface was to provide users with a consistent, familiar keyboard interface and in-line field verification while adding graphical features, mouse navigation and Windows functionality. The result is a Windows interface that maintains its ties to the existing product while providing a new look and feel.

In order to make use of the existing OSAS code, graphical events must be captured, interpreted, and simulated within the existing OSAS code structure.

So that your applications maintain a standard Windows interface, you must take great pains to assure that the rules of graphical interfaces are adhered to wherever possible as you convert your programs to graphical.

Building the .BRC File

All functions that have a screen in character mode must have a graphical version of that screen. All standard OSAS functions have an associated .BRC (Binary Resource file) file that contains one or more screens to use with the function. These files are stored in the **/osas/gui/** directory, but are installed only if your CD activation key grants you access to the GUI version of OSAS.

Naming Conventions

In naming these resource files, use the following convention:

1. If the program has only one screen, such as a report pick screen or simple maintenance screen, use the name of the program and add the extension **.BRC** (for example, the resource file for ARPRA would be ARPRA.BRC).
2. If the program is part of a chain of programs working together for a common task (for example, APACV, APACV1, APACV2, and APACV3, which all perform maintenance on vendors), name the resource file after the main part of the program name (in the example, use APACV.BRC). Within the resource file, add as many screens as the character version uses for the task.
3. If a single program is used for multiple tasks (such as for both maintenance and inquiry), use one resource file with 2 screens—one screen for the inquiry function and the other for the maintenance function.

All OSAS resource files are created using a combination of the BASIS utility ResBuilder (which can be installed with the Basic Developer's Kit) and a program in the Software Development Utilities that helps to create a resource file from a character screen layout.

Graphical Control Standards

In the charts below you'll see the controls used in OSAS resource files and the standards for using and maintaining them:

Forms

Forms in a .BRC file must start with control ID 1 and increase by 1 for each form.

Property	Value
Title	<title of character screen>
Name	frm<screen description> (for example, frmCustomers)
X Position	15
Y Position	45
Width	600
Height	400 ¹
Fore Color	<System Default>
Back Color	<System Default>
Default Font	MS Sans Serif, Regular, 8
Current Units	Pixels
Flags	Close Box, Enter as Tab, Invisible, Keyboard Navigation, Minimizable
Event Mask	Activate, Control focus, Key press, List click
Menu Status	<System Default>
Has status bar	Yes

¹Height may be greater if necessary, but should be no more than 425 pixels in order to maintain 640x480 compatibility when the status bar and menu bar are added.

Status Bar

Status bars exist on most forms. Status bars are not generally used on forms that appear over another form, but work together with the other form to accomplish a specific task. However, if a form appears over another form, but is a separate process (such as **F6** Maintenance forms, **Shift+F2** Information forms, and so on), then it will have its own status bar.

When you add a form manually using ResBuilder, and you enter **Yes** in the **Has Status Bar** form property, the status bar is added to the form automatically.

Property	Value
Control ID	100
Name	Status Bar
Text	<leave this field blank>
Font	MS Sans Serif, Regular, 8
Justification	<System Default>

Static Text Controls

Static text controls are used in two ways in OSAS. Field prompts are displayed in static text controls, as are field values that are *display only* (the value cannot be changed by the user).

Property	Value
Control ID	For field prompts, <1000, increasing by 10> For display-only fields - <2000, increasing by 10> ¹
Name	for field prompts - lbl<field name> (for example, lblCustID) for display-only fields - txt<field name> (for example, txtCustID)
Text	<same text as the character-mode prompts> (may also be populated through code)
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²

Property	Value
Height	15
Fore Color	<System Default>
Back Color	<System Default>
Font	for field prompts - MS Sans Serif, Regular, 8 for display-only fields - MS Sans Serif, Regular, 8, Bold
Justification	<System Default>

¹Standard OSAS screens have text and label fields numbered in increments of 10. This allows you room to add other controls between standard controls if necessary.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

INPUT Controls

INPUT controls are field entry controls. Use these controls wherever the user needs to enter alphanumeric data (with the exception of list fields and yes/no fields, which are handled with other controls).

Property	Value
Control ID	<4000, increasing by 10> ¹
Name	txt<field name> (for example, txtCustID)
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²
Height	18
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ³
Client Edge	checked
Mask	see Using Masks later in this section
Max Length	<number of characters> (same as X4 in text mode)

¹Standard OSAS screens have INPUTE controls numbered in increments of 10. This allows you room to add other controls between standard controls if necessary.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

³Refer to Disabling Fields later in this section.

INPUTN Controls

INPUTN controls are numeric entry controls. Use these controls wherever the user needs to enter numeric data (with the exception of fields that are used for selection a print/sort/pick type option, which are handled with other controls).

Property	Value
Control ID	<4000, increasing by 10> ¹
Name	txt<field name> (for example, txtSalesYTD)
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²
Height	18
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ³
Client Edge	checked
Mask	see Using Masks later in this section

¹Standard OSAS screens have INPUTN controls numbered in increments of 10. This allows you room to add other controls between standard controls if necessary.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

³Refer to Disabling Fields later in this section.

List Button Controls

List Button controls provide a list of choices from which a selection is required. Use these fields wherever the user needs to make a choice between two or more options (with the exception of Yes/No selections, which are handled by other controls).

Property	Value
Control ID	<4000, increasing by 10> ¹
Name	cbo<field name> (for example, cboCustType)
Initial Contents	<choice1>\n<choice2>\n<choice3>... (type \n between choices)
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²
Height	<depends on number of selections> ³
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ⁴
Client Edge	checked
Set Height Mode	<System Default>

¹Standard OSAS screens have list buttons numbered in increments of 10. This allows you room to add other controls between standard controls if necessary.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

³Make the height large enough to display all options when the list is displayed. If you make the control larger than necessary, the extra space will not be displayed. As a guide, a height of 40 will work for one option, 60 will work for two options, 80 will work for three options, and so on.

⁴Refer to Disabling Fields later in this section.

Check Box Controls

Check Box controls are used when the user must choose either **Yes** or **No**.

Property	Value
Control ID	<4000, increasing by 10> ¹
Name	chk<field name> (for example, chkGenlInfo)
Text	<same text as the character-mode prompt>
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²
Height	15
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ³
Client Edge	unchecked
Justification	<System Default>
Text Left	checked

¹Standard OSAS screens have check boxes numbered in increments of 10. This allows you room to add other controls between standard controls if necessary.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

³Refer to Disabling Fields later in this section.

Radio Button Controls

Radio Button controls are used to force a choice between several options. Use radio buttons wherever the user would enter a numeric value between 1 and n from a list of n choices (such as with sort options on report pick screens).

Property	Value
Control ID	<4000, increasing by 1> ¹

Property	Value
Name	rdo<field name> (for example, rdoSortBy)
Text	<same text as the character-based prompt> (Do not include the number in front of the option in the character screen.)
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²
Height	15
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	unchecked ³
Client Edge	unchecked
In Group	checked
Justification	<System Default>
Button Group	<any unique value> (Buttons with the same button group number will act together; when one is checked, the other button(s) will uncheck.)

¹Radio buttons must always be in increments of 1. The group box that should surround all radio groups are allocated in increments of 10, and the radio buttons increase from the group box number.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

³Refer to Disabling Fields later in this section.

Button Controls

Standard Button controls are used where command bar options exist on the equivalent character screen. These are normally found on HLE screens and on inquiry screens, where a list of commands appears at the bottom of the screen.

Property	Value
Control ID	<9000, increasing by 10> ¹
Name	btn<field name> (for example, btnFirst)

Property	Value
Text	<same text as the character-based command> (Use an ampersand [&] in front of the letter that activates the command.)
X Position	<position> ²
Y Position	<position> ²
Width	<button width> ²
Height	20
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ³
Client Edge	unchecked
Justification	<System Default>

¹Standard OSAS screens have buttons numbered in increments of 10. This allows you room to add other button controls between standard controls if necessary.

²Buttons are usually positioned at the bottom of the screen, equally spaced, equally sized, and horizontally lined up with each other. If one button requires more space because of the text needed for its command, that button should be larger than the other buttons only if space prohibits equal sizing. In this case enlarge the affected button and leave the other buttons equally sized.

³Refer to Disabling Fields later in this section.

Group Boxes

Group Boxes are used for grouping radio buttons and command buttons. They are also used to divide a form into multiple sections, in the same manner as the text-based screens are divided by boxes.

Property	Value
Control ID	for radio groups, use <4000, increasing by 10> ¹ for all other boxes, use <5000, increasing by 10> ¹
Name	grp<field description> (for example, grpSortBy)
Text	for radio groups, use text that describes the radio group for all other boxes, use no text

Property	Value
X Position	see Group Box Spacing below
Y Position	see Group Box Spacing below
Width	see Group Box Spacing below
Height	see Group Box Spacing below
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	unchecked
Client Edge	unchecked

¹Group boxes around radio buttons must be in sequence with the other controls (in increments of 10). The buttons inside the group should then be incremented by 1 from the group box (for example, a Sort By group box with 4 selections may be numbered 4050 for the group box, and the radio buttons numbered 4051, 4052, 4053, and 4054).

Group Box Spacing

Group Boxes can be lined up and spaced out to any desired setting. However, OSAS standard screens are laid out following a set of rules. For radio groups, follow these standards:

1. If a box contains 2 radio buttons, set the height of the group box to 59; for each additional radio button, add 17 pixels to the height. For the width, use whatever looks appropriate. Try to center the radio buttons within the group box, based on the longest button text.
2. Use a Y position 17 pixels higher than the first radio button.

For command bar group boxes, follow these standards:

1. Set the height of the group box to 35, and the width of the group box to the exact width of the entire form.
2. Use a Y position 11 pixels higher than the top of the buttons.

For group boxes that are used to divide the screen, follow these standards:

1. If boxes must start on the left-most part of the screen, use x-position 0. If the boxes must end at the right-most part of the screen, be sure that the x-position plus the width is equal to the entire width of the form.
2. For boxes that need to start at the top of the form, don't set the Y position lower than 27. This provides enough room for the tool bar on the form.
3. When adding multiple group boxes that are stacked vertically, set the y-position of the lower group box using this formula:

$$\begin{array}{r} \text{y-position of the upper group box} \\ + \text{ height of the upper group box} \\ - 3 \end{array}$$

This formula provides adequate vertical spacing between group boxes.

4. When adding multiple group boxes that are arranged horizontally, set the x-position of the right-most group box using this formula

$$\begin{array}{r} \text{x-position of the right-most group box} \\ + \text{ width of the right-most group box} \\ + 1 \end{array}$$

This formula provides adequate horizontal spacing between group boxes.

Tool Buttons for Scroll Regions

Tool buttons are used to control scrolling (HLE) regions within a graphical screen. Tool buttons include images that imply that the user can go to the top or bottom of a scroll region, or can scroll through pages and lines of entries.

The tool buttons are arranged in two groups of three in relation to the associated grid. The upper group consists of the buttons for **Home**, **Page Up**, and **Up** and is aligned with the top of the grid; while the lower group consists of the buttons for **End**, **Page Down**, and **Down** and is aligned with the bottom of the grid.

Property	Value
Control ID	<8001-6>

Property	Value
Name	8001 - btnHome 8002 - btnPageUp 8003 - btnUp 8004 - btnDown 8005 - btnPageDown 8006 - End
Text	<none>
X Position	<next to, but not overlapping, the right edge of grid>
Y Position	<in groups of 3 near top and bottom of grid>
Width	15
Height	18
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ¹
Client Edge	unchecked
Face Type	Bitmap
Bitmap File	8001 - OSHLEHM.BMP 8002 - OSHLEPGU.BMP 8003 - OSHLEUP.BMP 8004 - OSHLEDN.BMP 8005 - OSHLEPGD.BMP 8006 - OSHLEEND.BMP

¹Refer to Disabling Fields later in this section.

Tool buttons are also used on tool bars. Tool buttons for tool bars, however, are built by GENGUFNC.PUB and must never be built within a resource file.

Grids

Grids are used for scrolling text (HLE) regions. The user is allowed to scroll up and down through a grid to look at multiple pages of data.

Property	Value
Control ID	7000 (if more than one grid exists on the screen, increment by 10)
Name	grd<grid name>
Text	<blank>
X Position	<position> ¹

Property	Value
Y Position	<position> ¹
Width	<width> ¹
Height	<height> ¹
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ²
Client Edge	checked
Face Type	Bitmap
Num rows	<total number of rows to be displayed at one time (LN[2])>
Row height	17
Num columns	<total number of columns to be displayed in scroll region>
Column prop	<see Calculating Column Proportion below>
Col lines, Row lines	checked
Max cols	<same as value in Num columns>

¹You must visually line up each grid according to where the scroll region exists on the character screen.

²Refer to Disabling Fields later in this section.

Calculating Grid Column Proportions

Once you add a grid to the screen, you must determine how many columns will appear in the grid. You can determine the number the number of columns by examining the character screen. Set the **Num Columns** property to this value.

Next, you must decide how wide to make each column. You can try to estimate how wide each column should be and then adjust from there. The total width of all columns must not exceed the value in the **Width** property for the grid control. You can also use this chart to approximate the width of each column:

No. of Characters	Multiplier	Extended
1	17	17
2	14	28
3	13	39
4	13	52
5	13	65
6	13	78
7	11	77
8	11	88
9	11	99
10	12	120
11	10	110
12	10	120
13	7.5	97.5
14+	7.5	

For example, if a column is 15 characters wide, determine the width of the column by multiplying 15 by the multiplier of 7.5 to get 112.5 which, when rounded up, becomes 113.

Tool Buttons for F2 Inquiry

Tool buttons are placed to the right of certain INPUTE and INPUTN controls, and act as a shortcut to the F2 Inquiry window for the corresponding controls.

Property	Value
Control ID	<control ID of the input control + 10000> (for example, 14010).

Property	Value
Name	btnInq<input field name> (for example, btnInqCustID)
X Position	<x-position of input control + width of the input control -3>
Y Position	<same as y-position of INPUTE/INPUTN control>
Width	16
Height	18
Fore Color	<System Default>
Back Color	<System Default>
Font	MS Sans Serif, Regular, 8
Disabled	checked ¹
Client Edge	checked
Face Type	Bitmap
Bitmap File	OSINQUERY.BMP

¹Refer to Disabling Fields later in this section.

INPUTE/Grid Usage for Last Field

On each form that has any type of input, you must add a control with the ID 4999. This control must be either a grid or an INPUTE control. This control handles the final focus events when the user enters through an entire form.

For forms where a series of input fields are used, use an INPUTE control. For forms where you must prompt the user for single-key input (ex. setting X3=8 or X3=9), use a grid instead.

Property	Value
Control ID	4999
Name	txtEndField or grdEndField
X Position	0
Y Position	0
Width	0

Property	Value
Height	0
Fore Color	<System Default>
Back Color	<System Default>
Font	<System Default>
Disabled	unchecked
Client Edge	unchecked
Max Length	255 (for INPUTE controls only)
Num Rows	1 (for Grid controls only)
Row Height	20 (for Grid controls only)
Num Columns	1 (for Grid controls only)
Max Columns	1 (for Grid controls only)

Other Resource Standards

Whether you are designing new screens or modifying existing ones, keep these standards in mind:

1. All controls dealing with user input start at control ID 4000 and increase by 10. The control numbers coincide with the order that the fields are being used within the existing program. The increments of 10 allow you room to add controls between the existing fields. If you add fields, the control ID must fall between the control IDs that already exist.
2. All programs that work with OSAS must use the form flags and event masks as laid out in these standards. Using other flags and event masks may result in unpredictable behavior.

Other Control Standards

OSAS also uses controls that are defined by systems programs and that are not stored in resource files:

Control ID	Usage
20000-200130	Commands menu (Inquiry, Maintenance, etc.)
20050-20053	Edit menu (Copy, Paste, Undo)
20100-20105	Modes menu (Verify Exit, Bell, Quick, etc.)
20300-20304	Help menu (About OSAS, Command Help, etc.)
20200-202xx	Other Commands menu (Calculator, Call BBx program, etc.)
20500-20509	Scroll region menu (Home, End, Page Up, Delete, etc.)
20550-2055x	Information menu (Shift-F2 menu in character mode)
20600+	Inventory Shift-function-key menu, Var usage
21000-21200	Tool Bar - Several types of controls make up the tool bar

Calculation of Field Sizes

Use this table to determine the pixel width of input controls in OSAS screens:

No. of Characters	Multiplier	Extended
1	17	17
2	14	28
3	13	39
4	13	52
5	13	65
6	13	78
7	11	77
8	11	88
9	11	99
10	12	120
11	10	110

No. of Characters	Multiplier	Extended
12	10	120
13	7.5	97.5
14+	7.5	

For INPUTE or INPUTN fields, determine how wide to make an entry field by applying the table value corresponding to the character field length. For example, for a field with a character length of 15, calculate the length by multiplying 15 by the multiplier of 7.5 for a result of 112.5 which, when rounded up, becomes 113.

For static text controls, either estimate the space needed by sight, or use the formula above. Then add a few pixels to account for any scaling that may happen if the user turns scaling on.

Using Masks Within Resource Files

You can define the mask that an alphanumeric or numeric field should use within a resource file. You can change these masks from within the code, if necessary.

For an INPUTN control, specify the mask you want to use for the input. Use ‘#’ and ‘0’ to represent numeric characters. If you do not want a ‘0’ to appear if it is entered, use the ‘#’.

For an INPUTE field, the following chart shows the possible values for the mask:

Mask	Function
X	Any printable character
a	Any alphabetic character
A	Any alphabetic character. Converts letters to uppercase.
0	Any digit
U	Any digit, alphabetic, space, or punctuation. Converts letters to uppercase.
z	Any digit or alphabetic character
Z	Any digit or alphabetic character. Converts letters to uppercase.

Creating Resource Files from OSAS Standard Code

The Advanced Software Development Utilities contains two programs that will generate resource files automatically from existing OSAS programs. The programs are called GENINPUT.PUB and SDBLDARC. You must run these utilities on a Windows-based system.

This special GENINPUT.PUB has been modified to record field sizes, types, lengths, and so on in a text file built to PRO/5 .ARC (ASCII Resource) file standards. (Refer to the BASIS manuals on ASCII Resource Files and RESCOMP.EXE for more information on the format of these files.)

SDBLDARC examines the 7500 section of code (character prompts) in the current workspace program and creates an .ARC file with this information. Finally, SDBLDARC merges the two .ARC files together into one and runs RESCOMP.EXE. RESCOMP.EXE is a BASIS utility (which ships with Resource Manager) that uses the combined .ARC file to create a .BRC file.

If your original character-based program has been developed in accordance with OSAS standards, the resulting .BRC file will require only minimal cleanup work to work with your programs.

Preparing and Installing the Utilities

Follow these steps before you install and use the utilities:

1. Install your Developer's Kit. This includes both the Software Development Utilities and the BASIS utilities, so that you can run the BASIS ResBuilder utility. The installation process renames the existing GENINPUT.PUB and installs the Developer's Version, which can be used to create graphical screens.
2. Move the BASIS utility RESCOMP.EXE and the file STDINCL.H (both of which should already exist in the progdm directory of your OSAS v6.5 install) into a Windows system path, such as **!windows!command!l**. This is required because RESCOMP.EXE is called through an operating system command shell from within the application directories, so Windows must be able to find the programs.
3. Install your character application under OSAS 6.5.

Before Running the Utilities

The graphical screen creation utilities are designed to speed up the process of creating graphical resource files for use with OSAS programs. The utilities create a best-guess resource file, and you may need to clean up the resulting file a little before you use it with your programs.

However, you may find it more efficient to evaluate your programs and make certain changes before you run the utilities. Here are some suggestions for changes you can make before the creation to make the resulting cleanup process easier:

1. Check the 7500 section of the original program and remove spaces within PRINT statements, such as:

```
7500 PROMPTS:
7510 PRINT @(2,2),"Customer ID      From"
7520 PRINT @(2,3),"                Thru"
```

The spaces in these examples will cause the **From** and **Thru** prompts to be misaligned in the resulting resource file. Fix this problem with the following code:

```
7500 PROMPTS:
7510 PRINT @(2,2),"Customer ID",@(17),"From"
7520 PRINT @(17,3),"Thru"
```

2. Break up long prompts into separate prompts. In the above example, notice that the **Customer ID** and **From** prompts are **split** apart. Another example of this is as follows:

```
7500 PROMPTS:
7510 PRINT @(2,2),"Pick Customer ID From"
```

Change this code to read as follows:

```
7500 PROMPTS:
7510 PRINT @(2,2),"Pick",@(7),"Customer ID",@(19),"From"
```

This will allow your prompts to properly line up with each other.

3. Remove variables from positions in PRINT statements. For example:

```
7500 PROMPTS:
7510 PRINT @(3,YPOS),"Pick",@(7),"Customer ID",@(19),"From"
7510 PRINT @(19,YPOS+1),"From"
```


The screen cannot be created correctly from this code because, at the time the resource file is built, the value of YPOS is unknown. Temporarily replace this code with the actual position of the field to allow SDBLDARC to properly build the file for you. Then change your code back, if necessary, when done.

4. Repair programs that use this coding method for Yes/No fields:

```
1845 LET X0=12,X1=50,X3=5,X4=3,X$=YESNO$,X1$="YESNO "
```

The resource creation program will build a list button for the Yes/No prompt in this example, which is not the proper GUI control. The proper coding method for Yes/No fields is to remove or clear the values in X1\$ variable, as in the following example:

```
1845 LET X0=12,X1=50,X3=5,X4=3,X$=YESNO$,X1$=""
```

When Yes/No fields are coded in this way, the resource creation program will build the proper check box, instead of the improper list button.

Using the Utilities

To build graphical screens, follow these steps:

1. From the OSAS menu, run the function for which you want to create a graphical screen.
2. At the first field on the screen, press **Ctrl+A**. The program name appears in the upper right corner of your screen, with the word **ON** after it.
3. Enter through each field on the screen, making sure that you have valid input in each field as necessary. As you enter through each field, the special GENINPUT.PUB program records field information. Do not use any commands (including the **Up** command), as fields may be recorded more than once or not at all. In addition, if you see any error messages that are generated in your program (such as 'xxxxxx is not on file' or 'Thru value must be greater than from value'), you will also have duplicate field information recorded.
4. When you get to the last field on the screen, **do not** enter through the field. Instead, press **Ctrl+A** again to terminate the field recording and to run the SDBLDARC program.

5. When the **Build BRC Now (Y/N)?** prompt appears, enter **Y** to complete the graphical screen generation. When you do, the SDBLDARC program examines the 7500 section of the program currently loaded and builds the prompts into another .ARC file. The program then combines the field and prompt information in a single .ARC file and calls the RESCOMP.PUB program to build the resource file.

The new resource files are created in the application programs directory. They are not created in the graphic files directory (such as /osas/gui) so that existing resource files are not accidentally overwritten. After the resource file creation is complete, you must move the file to the graphical files directory manually.

Cleaning up Resource Files

Once the resource file is created, use the BASIS ResBuilder utility to open and edit the file. The utility provides a graphical tool to manipulate the forms and controls in the resource file.

Use ResBuilder first to account for all controls that should be included on the form. Some fields may not have been included, because they appear only when certain conditions exist (such as application interfaces). These fields must be included in the resource file. They are usually hidden or shown by the code within the program where necessary (see Converting Your Programs below for more information on hiding and showing controls). Some fields must be added manually as well.

Follow these steps to check for changes necessary in the resource file:

1. Check for prompts and data entry that are displayed with a variable or based on an option switch. The following code is taken from ARPRH (Detail History Report)

```
7660 IF KIT THEN PRINT @(54,18),"Kit Detail",
```

In this example, you must have the option for keeping kit history turned on through Sales Order. This prompt will always be built into the resource file, but you may notice the input field missing. To alleviate this problem, make sure you have Bills of Material/Kitting installed and the option set. You should also create this screen using Sales Order rather than Accounts Receivable (which doesn't have an option for kitting).

Check also for missing **F2 Inquiry** tool buttons. Here is an example of code from an Accounts Receivable program:

```
1845 LET X0=I+3,X1=65,X3=0,X4=12,X$=TD1$(25+(I-1)*12,12),
1845:GUI_ID=4090+(I*10),PROGID$="GLACGFM.PUB"
1850 IF O2$(1,1)="1" THEN LET X3=13,X$=CVS(X$,2),X4=LEN(X$),
1850:WINDID$="ARENTBW2",SELVAL$=CCID$+FILL(12)
```

The settings on line 1850 will be used only if you are interfaced to GL. If GL is not interfaced, no F2 button will be built next to the field of input.

Check your programs closely for option-related code before you create the graphical screens. Finding and fixing option-related prompts, fields, and inquiry buttons in advance can save you time spent repairing the resource files later.

2. If the character screen is divided into sections, you must manually add group boxes to divide the graphical screen into sections. The creation program will not create the boxes for you.
3. Because selection fields (**Sort By**, **Print By**, and the like) are coded as numeric fields in the character-based code, the resource creation program will not change them to radio button groups. You must add the radio buttons and groups manually.
4. The utility creates a separate BRC file for each screen; however, the OSAS standard is to keep all screens for a program chain in the same BRC file for performance reasons. Therefore, after you create all screens for a function, use ResBuilder to copy and paste these screens into one resource file.
5. Horizontal Line Entry (HLE) screens are far more complex in layout. To create screens with a combination of regular entry and HLE, record the regular entry portion first, clean up your BRC as necessary, and then manually add the grid (or grids), scroll buttons, and command buttons using ResBuilder.

Other Resource Cleanup Issues

You must also make some additional changes to the resource files for internal standards reasons:

1. Add the proper name to each control in the resource file. The naming convention is to use an initial prefix that specifies the type of control (such as **txt** or **cbo**) and then a description of the control. See “Graphical Control Standards” on page 4-22 for specific naming conventions. Failure to add the names will not affect your programs, but may cause confusion when working with the resource file at a later date.
2. The creation program builds the prompt for a Yes/No check box field independently from the check box control. You should add the text as a label in the check box control, resize the check box control if necessary, and then remove the static text control. Adding the prompt to the check box control will cause the system to display a dashed line around the prompt when the check box control gains focus.
3. Verify that the programs use the control numbering standards layed out above to avoid confusion and to enable possible future modifications.

Disabling Fields

In several graphical OSAS functions, many fields are disabled. This is generally done in maintenance, inquiry, and transactions screens to prevent the user from moving into other portions of the screen without entering required information.

In general, OSAS functions follow these rules for disabling fields:

1. In maintenance functions, enable the key field and disable all other fields until the key is entered. Then disable the key field and enable the remaining fields.
2. In inquiry functions, enable the field or fields that must be entered before data is displayed and the command bar becomes available. After the data is entered, disable the selection fields and enable the remaining fields.
3. In transaction functions, enable fields in groups to restrict movement as necessary. For example, you may enable only the key field (such as customer ID, vendor ID, or the like) while disabling all other fields. When the key field is entered, enable the remaining fields, keeping the key fields available because the user can arrow back to change them. In editing mode, disable fields that cannot be changed.

Set the enabled/disabled properties of the controls in the resource file to their initial values to avoid the flicker that occurs when fields are displayed and then disabled.

Converting Programs

After you convert and clean up the resource file for a function, you must make the changes in the programs themselves to handle both character and graphical modes. OSAS uses the same code for both interfaces.

As you convert the programs, you may need to make adjustments to the resource file. If you followed the steps above to clean up the resource file, you should not need to add input fields, but you may need to add other labels where necessary.

You must add or change several areas of code to make your programs run in both graphical and character mode. The order in which you make the changes does not matter; however, you must make all the changes before the program runs successfully.

Merge New General Routines

Before you begin, use the BASIS _update utility and the GENLIB.61 library file to update the common routines. Starting with version 6.1, the routines GENINPUT, GENLINE, GENERERROR, GENLOCK, and GENNUM have all changed.

Convert Screen Prompts (7500)

The Prompts section of code displays field labels on a character screen after boxes are drawn on the screen by the Box (7900) section of code. Here is a sample of code from a typical Prompts section:

```
7500 PROMPTS:
7502 GOSUB BOX
7504 PRINT `SB`,STBL("FTXT"),
7520 PRINT @(3,2),"Pick Customer ID      From",@(52,2),"Print By:",
7530 PRINT @(23,3),"Thru",@(54,3),"1.  Customer ID",
7540 PRINT @(8,4),"Sales Rep 1      From",@(54,4),"2.  Sales Rep ID",
.
.
7660 IF KIT THEN PRINT @(54,18),"Kit Detail",
7692 RETURN
```

As you can see, several changes were required before creating the BRC based on the suggestions above. The code should look something like this:

```
7500 PROMPTS:
7502 GOSUB BOX
7504 PRINT 'SB',STBL("FTXT"),
7520 PRINT @(3,2),"Pick Customer ID",@(23,2),"From",@(52,2),"Print By:",
7530 PRINT @(23,3),"Thru",@(54,3),"1. Customer ID",
7540 PRINT @(8,4),"Sales Rep 1",@(23,4),"From",@(54,4),"2. Sales Rep ID",
.
.
7660 IF KIT THEN PRINT @(54,18),"Kit Detail",
7692 RETURN
```

After you make these changes and properly generate the graphical screen, add a line to this section of code that branches to GUI_PROMPTS, a section of code that retrieves the graphical screen from the resource file. You must retrieve the screen before doing any screen paints. Here is a sample of the line of code:

```
7501 IF GUI THEN GOTO GUI_PROMPTS
```

The variable GUI is set by the menu programs. It is set to 1 if the functions are run in graphical mode, or 0 if not. The line should be positioned in the code block so that if GUI=1, the program skips the character prompt display and branches to GUI_PROMPTS.

Add Graphical Prompts (17500)

Although the code is different, the graphical GUI_PROMPTS section plays the same role as the character-based PROMPTS section: it displays the screen and paints the screen prompts. GUI_PROMPTS also displays the data controls on the form as well. Here is a sample of the GUI_PROMPTS section:

```
17500 REM 17500 "----- GUI Prompts and Box"
17510 GUI_PROMPTS:
17520 LET
OSGUI.SCRNID$="ARPRH.BRC",OSGUI.WIND_ID[1]=1,OSGUI.CONTEXT_POINTER=1
17530 LET MSG$=""; CALL "GENGURES.PUB",FILE[ALL],OSGUI$,MSG$; IF
17530: CVS(MSG$,3)<>" THEN LET X9$=MSG$,O7=1; GOSUB GENERERROR; GOTO QUIT
17540 CALL "GENGUFNC.PUB::BUILD_MENU_STRINGS",MENUHLP$,MENU1$
17545 IF O2$(2,1)="1" THEN CALL
"GENGUFNC.PUB::BUILD_ITEM_LOOKUP_MENU_STRINGS",
17545:MENUIITEM$
17550 LET MENU1$=MENU1$+MENUIITEM$+MENUHLP$; PRINT
(OSGUI.CHAN)'SETMENU',MENU1$;
17550:LET MENU1$="",MENUHLP$="",MENUIITEM$=""
17555 IF KIT THEN PRINT (OSGUI.CHAN)'SHOW'(4220)
17560 IF O2$(22,1)<>"1" THEN PRINT (OSGUI.CHAN)'DISABLE'(4210)
17565 IF O2$(2,1)="1" THEN PRINT (OSGUI.CHAN)'SHOW'(14040,14050),
17565:'DISABLE'(-20600)
```

```
17570 PRINT (OSGUI.CHAN) 'TITLE' (0,TITLE$), 'SHOW' (0)
17575 RETURN
```

Refer to “GUI Screen Prompts” on page 2-40 for more information on specific types of GUI_PROMPTS sections.

Convert Screen Data (7700)

The DATA section of code is used to display data in the fields on the character screen. Here is a sample of a typical DATA section:

```
7700 DATA:
7704 PRINT 'CF', 'SF',
7710 PRINT @(29,2),N1$; REM Cust ID
7720 PRINT @(29,3),N2$
.
.
7860 PRINT @(8,20),N5:"#"
7890 RETURN
```

In the same way as the PROMPTS section, you must add a line that skips the character data paint routine and instead branches to a graphical data painting routine. Here is a sample of the line to add to this program:

```
7702 IF GUI THEN GOTO GUI_DATA
```

In graphical mode, the line skips to the GUI_DATA section. In text mode, the data is painted as usual.

Add Graphical Screen Data (17700)

Like the prompt sections, the GUI_DATA section plays the same role as DATA: it displays the data on the screen. Here is a typical GUI_DATA section:

```
17700 REM 17700,5 "----- GUI Data"
17710 GUI_DATA:
17715 PRINT (OSGUI.CHAN) 'TITLE' (4000,N1$)
17720 PRINT (OSGUI.CHAN) 'TITLE' (4010,N2$)
17725 PRINT (OSGUI.CHAN) 'TITLE' (4020,N3$)
17730 PRINT (OSGUI.CHAN) 'TITLE' (4030,N4$)
17735 PRINT (OSGUI.CHAN) 'TITLE' (4040,N5$)
17740 PRINT (OSGUI.CHAN) 'TITLE' (4050,N6$)
17745 PRINT (OSGUI.CHAN) 'TITLE' (4060,N7$)
17750 PRINT (OSGUI.CHAN) 'TITLE' (4070,N8$)
17755 PRINT (OSGUI.CHAN) 'TITLE' (4080,FND$(N1))
17760 PRINT (OSGUI.CHAN) 'TITLE' (4090,FND$(N2)); LET TMP=FNCHECK(4150,FNB$(1))
17765 PRINT (OSGUI.CHAN) 'TITLE' (4100,INV_FROM$); LET
TMP=FNCHECK(4160,FNB$(2))
17770 PRINT (OSGUI.CHAN) 'TITLE' (4110,INV_THRU$); LET
TMP=FNCHECK(4170,FNB$(3))
```

```

17775 LET TMP=FNCHECK(4180,FNB$(4))
17780 LET TMP=FNCHECK(4190,FNB$(5))
17785 LET TMP=FNCHECK(4200,FNB$(6))
17790 LET TMP=FNCHECK(4210,FNB$(7))
17795 LET TMP=FNCHECKRADIO(N3,"0412104122"),TMP=FNCHECKRADIO(N5,"0413104132")
17795:,TMP=FNCHECKRADIO(N4,"04141041420414304144")
17800 IF KIT THEN LET TMP=FNCHECK(4220,KIT_DETAIL$)
17805 RETURN

```

Refer to “GUI Data Display” on page 2-46 for more information on the GUI_DATA section.

Add GENNUM, GENPHONE

In programs that display or allow entry of numeric data or phone numbers, add and implement the GENNUM and GENPHONE sections of code.

Add GENCHECK, GENCHECK_RADIO, and GENLIST

As you add the GUI_DATA section of code, add and implement the GENCHECK, GENCHECK_RADIO, and/or GENLIST sections as needed. Use GENCHECK for check boxes on the form, GENCHECK_RADIO for radio buttons, and GENLIST for list boxes.

Add GUI_ID

On each line where variables are assigned before a CALL to GENINPUT.PUB, you must assign a graphical control ID to the field. Add the variable GUI_ID to the end of the line where the other GENINPUT variables are assigned (X0, X1, X3, X4, and so on). The GUI_ID variable is also set within HLE program before CALLing GENLINE to tell GENLINE which grid control to use.

Add GUI_RADIO\$

If the program uses radio buttons, you must use the GUI_RADIO\$ variable to specify which buttons make up the radio group. This variable must contain the control ID of each radio button. Each control ID must be 5 characters in length. Here is a sample of the GUI_RADIO\$ assignment in ARPRH:

```

1730 LET
X0=17,X1=8,X3=1,X4=1,X1$="#",X=N3,GUI_ID=4120,GUI_RADIO$="0412104122"

```

In this example, radio buttons 4121 and 4122 below to this radio group. GUI_ID is set to the control ID of the group box.

Add CLEAR_GLOBAL

You must add the CLEAR_GLOBAL section to most programs that require user-interaction. Here is a sample of the CLEAR_GLOBAL section:

```
18200 REM 18200,5 "----- Clear GUI Global"
18205 CLEAR_GLOBAL:
18210 IF GUI THEN LET DUMMY$=STBL("!CLEAR","DESTID",ERR=18215)
18215 RETURN
```

When a user clicks to move from one field to field on a graphical screen, the ID of control on which the user clicked is stored in a global variable called DESTID. The global variable is used in GENGUINP.PUB to tell which field the user clicked on. GENGUINP.PUB then passes back the appropriate variable in Y\$: Y\$="U" if the user clicked on a field above the field with focus, or Y\$="E" if the user clicked on a field below the field with focus. (This is why the standards require the fields in the resource file to be numbered in increasing order.) When GENGUINP.PUB is CALLED with the same GUI_ID as the DESTID global, GENGUINP.PUB clears the global and handles input as usual.

The movement, then, of the field focus is handled automatically by GENGUINP.PUB, and requires no special program code.

However, it may be necessary to stop this movement at times. For example, at the end of the screen's processing (in the PROCEED or OUTPUT sections, for example), if an error message must be displayed, or if required information is not available for a field, you must use the CLEAR_GLOBAL section to clear the DESTID global to stop GENGUINP.PUB from automatically issuing the Y\$ commands.

The result of not including this call is an endless loop if you click on certain fields, because DESTID will contain a control ID until the program reaches that control or until the program specifically clears the variable. Here are some examples that illustrate the proper use of the CLEAR_GLOBAL routine:

```
1070 LET X0=3,X1=29,X3=0,X4=6,X$=N2$,WINDID$="ARCIQ W1",GUI_ID=4010
1080 GOSUB GENINPUT; REM "CUST # - THRU"
1085 IF Y$="U" THEN GOTO 1030
1090 LET N2$=X$+FILL(6-LEN(X$))
1110 IF N2$<N1$ AND N2$<>FILL(6) THEN LET X9$=FT$; GOSUB CLEAR_GLOBAL; GOSUB
1110:GENERERROR; GOTO 1070
```

In this code sample, the user must enter a customer ID in the **Customer Thru** field that is greater than the value in the **Customer From** field. If not, the program branches from 1110 back to 1070. In case the user enters a **From** value greater than the existing **Thru** value, and then clicks on a control after the **Thru** field, the program will use the CLEAR_GLOBAL subroutine to clear DESTID.

Here's another example:

```
1070 LET X0=2,X1=18,X3=0,X4=6,X$=C0$,WINDID$="ARACC W1",P0=0,GUI_ID=4000
1080 GOSUB GENINPUT; REM "CUSTOMER ID"
1100 IF Y$="U" THEN GOTO 1000
1110 LET X$=X$+FILL(6-LEN(X$))
1120 LET C0$=X$
1130 IF X$=FILL(6) THEN LET Y$=""; GOSUB CLEAR_GLOBAL; GOTO 1030
```

In this code sample, the user must enter an ID. If the user clicks on a field below the ID field without entering an ID, the CLEAR_GLOBAL subroutine prevents a looping issue if the user clicks on any other field.

There may also be places where you need to jump around controls based on data entered in other fields. This normally happens with check boxes and list buttons, as in this example:

```
1630 IF X$(1,1)="Y" THEN LET C4$(6,1)="1" ELSE LET C4$(6,1)="0"
1632 IF SAVEX$="NO" AND X$="YES" AND GUI THEN GOSUB CLEAR_GLOBAL ELSE IF
1632:SAVEX$="YES" AND X$="NO " AND GUI AND STBL("DESTID",ERR=1635)="4020" THEN
1632:GOSUB CLEAR_GLOBAL
1635 IF C4$(6,1)="0" THEN IF GUI THEN PRINT (OSGUI.CHAN)'DISABLE'(4020,14020)
1635:FI; GOTO CREDIT_LIMIT ELSE IF GUI THEN PRINT (OSGUI.CHAN)'ENABLE'
1635:(4020,14020)
```

This code sample checks the user's input in the **Finance Charge?** field. If the user is running in graphical mode and changes the answer from **YES** to **NO**, the program checks to see if the **Finance Charge Code** field gets focus; if it does, the program clears DESTID to prevent an endless loop. If the user changes the answer from **NO** to **YES**, the program always clears DESTID so that the program will stop at the **Finance Charge Code** field.

Convert Speedos

In programs that display counters or *speedos* at the bottom of the screen, you must take special steps to convert the speedos to GUI. Search all programs for speedos, because speedos are used in many types of programs, including report printing programs and functions that do some type of processing that requires time to run. In graphical mode, speedos will appear on the status bar.

This sample prints a speedo on the status bar in graphical mode, but as a standard text message in text mode:

```
1002 IF DEVICE$ <> FID(0) THEN IF GUI THEN PRINT (OSGUI.CHAN) 'TITLE'
      (100, "Printing...") ELSE PRINT @(1,22), FILL(77), 'SB', @(2,22), "Printing...",
```

Convert Other Screen Updates

Look for other places in your program where you print data to the screen without using the DATA section of code. You must then add the appropriate code to your programs, and the appropriate fields in your resource file, to accomplish this. Here is a sample from a pre-6.1 version of ARENTLN2 that displays the shipped and backordered quantities after the ordered quantity and unit of measure were entered:

```
1660 PRINT 'SF', STBL("FTXT2"), @(11,9), FNNUM$(TD5[1], MQTY$),
1670 PRINT 'SF', STBL("FTXT2"), @(11,10), FNNUM$(TD5[2], MQTY$),
```

This code is changed as follows to work in graphical mode:

```
1660 IF GUI THEN PRINT (OSGUI.CHAN) 'TITLE' (4150, FNGUINUM$(TD5[1], MQTY$, 4150))
1660: ELSE PRINT 'SF', STBL("FTXT2"), @(11,9), FNNUM$(TD5[1], MQTY$),
1670 IF GUI THEN PRINT (OSGUI.CHAN) 'TITLE' (4160, FNGUINUM$(TD5[2], MQTY$, 4160))
1670: ELSE PRINT 'SF', STBL("FTXT2"), @(11,10), FNNUM$(TD5[2], MQTY$),
```

Pay particular attention to inquiry program types. In these types of programs, the user enters in a value on which to inquire, and a whole screen of data is presented. Because these fields are display only, they are not picked up by the graphical screen creation utility, and you must manually build these fields into your resource files.

This applies to any field where input from the user is not required. You must place the appropriate control (usually a static text control or a check box control) into your resource file manually.

Add Code to Disable and Enable Fields

As discussed earlier in this chapter, you must enable and disable fields at certain times in your programs.

For example, when you first enter the **Payment Methods** function in Accounts Receivable, the **Payment Method** field is enabled and the remaining fields are disabled to force entry of a valid payment method. When the payment method is entered, the **Payment Method** field is disabled and the remaining fields are enabled. When you return to the payment method field (after the **Abandon** or **Proceed** commands), the **Payment Method** field is again enabled and the remaining fields are again disabled.

This code sample handles both enabling and disabling of controls:

```
18000 REM 18000,5 "----- Disable Fields"
18005 DISABLE_FIELDS:
18010 IF GUI THEN PRINT (OSGUI.CHAN)'HIDE'(1040,4040,14040),
'ENABLE'(4000,14000), 'DISABLE'(4010,4020,4030,4050,14050,4060,4070,4080,4090)
18015 RETURN

18100 REM 18000,5 "----- Enable Fields"
18105 ENABLE_FIELDS:
18110 IF GUI=0 THEN GOTO 18125
18115 IF P1<=3 AND P1>0 THEN PRINT (OSGUI.CHAN)'SHOW'(1040,4040), 'ENABLE'(40
18115:40);IF P3WIND$<>" THEN PRINT (OSGUI.CHAN)'SHOW'(14040), 'ENABLE'(14040)
18115:ELSE PRINT (OSGUI.CHAN)'HIDE'(14040) FI ELSE PRINT (OSGUI.CHAN)'HIDE'
18115:(1040,4040)
18120 PRINT (OSGUI.CHAN)'DISABLE'(4000,14000),
18120:'ENABLE'(4010,4020,4030,4050,14050,4060,4070,4080,4090)
18122 IF (O2$(15,1)="1" AND P1<3) OR CUSTOK=0 THEN PRINT (OSGUI.CHAN)'DISABLE'
18122:(4050,14050)
18125 RETURN
```

The DISABLE_FIELDS section disables the **Payment Method** control and its corresponding **F2 Inquiry** tool button, and enables all other controls. This example also hides controls 1040, 4040, and 14040, which are the customer ID and bank ID and the associated F2 Inquiry tool button.

The ENABLE_FIELDS section disables the **Payment Method** control and its corresponding **F2 Inquiry** button, and enables the remaining fields. Then it checks the payment type to determine which other fields to show and enable, including the GL account, customer ID, or bank account ID and their **F2 Inquiry** buttons.

Add Code to Hide and Show Controls

Some programs have fields that appears and disappear depending on what is happening in the program. For example, most file maintenance programs contain a **Copy From** field that is not visible until you enter a new ID. The prompt and the field then appears on the screen. After you enter the **Copy From**, the field is removed from the screen.

The graphical equivalent for this technique is to include the control for the *hidden* field, and set the **Visible** property to invisible. Then when the field is needed, use the 'SHOW' mnemonic in the program to make the field visible. When the field is no longer needed, use the 'HIDE' mnemonic to remove the field from view.

You can decide to hide optional fields in the resource file, then show them in code, or have them visible in the resource file, then hide them in code. However, you should choose one method or the other for consistency.

If the field was not used when you used the graphical screen creation utility, you must add both the label and the input controls for the hidden field to the resource file. Because the field prompt is usually not built in the 7500 section, you will most likely have to add the label control in any case.

This code is from a pre-6.1 version of ARACC:

```
1310 PRINT 'SB',STBL("FTXT"),@(31,2),"Copy from"
1400 PRINT @(31,2),FILL(46)
```

To convert this to work in graphical mode, check for graphical mode, and then show and hide the controls when necessary:

```
1310 IF GUI THEN PRINT (OSGUI.CHAN)'SHOW'(1030,4010,14010) ELSE PRINT
1310:'SB',STBL("FTXT"),@(31,2),"Copy from"
1400 IF GUI THEN PRINT (OSGUI.CHAN)'HIDE'(1030,4010,14010) ELSE PRINT
1400: @(31,2),FILL(46)
```

Keep in mind that the user may use the **Abandon** command in the **Copy From** field, so similar logic is needed in the 8470 section to clear these prompts also.

You may find other examples of hidden and shown fields (such as the Batch ID fields in AR, SO, AP, and PO). In these cases, the code for hiding and showing the field resides in the GUI_PROMPTS section. Here is a sample from ARENTHDR:

```
17550 IF O2$(10,1)="1" THEN PRINT (OSGUI.CHAN)'SHOW'(1000,4000,14000)
```

This code checks the batching option, and if it is set, uses the 'SHOW' mnemonic on the label, the input field, and the **F2 Inquiry** tool button.

Convert CALLs to Public Programs

If you are converting a public program that runs in both character and graphical modes, and you specify variables in your CALL/ENTER lists, you must add the OSGUI\$ template to the CALL/ENTER list. OSGUI\$ is required because it is used by the general routines to determine the mode in which OSAS is running.

After the ENTER statement in the public program, add a line like this:

```
0060 IF OSGUI$<>" " THEN IF OSGUI.CHAN THEN LET GUI=1
```

While you could also just pass the variable GUI in your CALL/ENTER list, standard OSAS programs use the above method.

Add Code to Destroy or Hide Windows (Forms)

Many programs use more than one graphical screen. Transactions functions, some maintenance functions, some inquiry functions, and some commands (such as **Maintenance**) display new graphical windows over existing windows.

When the user finishes with the new window, the program must either 'HIDE' the window or 'DESTROY' it completely (much as the text version uses the 'POP' mnemonic).

First, determine whether you should 'HIDE' or 'DESTROY' the window. In general, if the window will likely be used more than once, you should 'HIDE' the window so it does not have to be loaded again later. If the window is used sparingly, 'DESTROY' the window to free resources.

This sample from ARACCFM.PUB demonstrates the 'DESTROY' mnemonic:

```
18400 REM 18400,5 "----- Destroy Window"
18405 DESTROY_GUI_WINDOW:
18410 IF GUI_POP THEN PRINT (OSGUI.CHAN)'DESTROY'(0) FI; LET OSGUI$=
18415:SAVEOSGUI$;PRINT (OSGUI.CHAN)'CONTEXT'
18415:(OSGUI.CUR_CONTEXT[OSGUI.CONTEXT_POINTER])
18415 RETURN
```

This code uses the variable GUI_POP, which is set by the GUI_PROMPTS section of code. If this variable is set, the 'DESTROY' mnemonic is used to remove the entire screen from memory. Because the window no longer exists, the OSGUI\$ template is then reset to its original value, as is the current context.

To hide the window, substitute the 'HIDE' mnemonic for the 'DESTROY' mnemonic in the previous example. In addition, you must also save the element of the GUI_WIND[x] array on which this window is open, so you can refer to it when you need to show the screen again.

You can also hide a window in the PROMPTS section. This code sample from ARACC hides the current window and then sets the context back to the initial Customer maintenance screen, if it has already been opened:

```
17500 REM 17500 "----- GUI Prompts and Box"
17510 GUI_PROMPTS:
17515 IF ARACC_POINTER THEN PRINT (OSGUI.CHAN)'HIDE'(0),
17515:'CONTEXT'(OSGUI.CUR_CONTEXT[ARACC_POINTER]); LET OSGUI.CONTEXT_POINTER=
17515:ARACC_POINTER; GOTO 17545
17520 LET OSGUI.SCRNID$="ARACC.BRC",OSGUI.WIND_ID[1]=1,
17520:OSGUI.CONTEXT_POINTER=-1,ARACC_POINTER=1
```

As a general rule, windows created by **F6 Maintenance** functions, **Shift+F2** Information functions, additional descriptions logic, and other *black-box* programs are destroyed. In a chain of programs in which the window is used repetitively, such as in file maintenance or daily work functions, windows should be hidden.

Add GUI_ID and GUI Buttons to HLE and Inquiry Programs

If you are working with Inquiry, HLE, or other functions where you use a command line to activate certain tasks, you must add button controls to your resource file, along with a grid control.

For HLE screens, the control ID of the grid must be 7000, 7010, and so on., depending on how many scroll regions are on the screen. For inquiry functions and other programs that use X3=9 keystroke commands, the control ID of the grid must be 4999.

When you convert these programs, you must set GUI_ID each time you CALL GENLINE, or when you set X3=9 before you CALL GENINPUT.

You must also use the `BUTTON_KEYS$[ALL]` array to tell the `GENLINE` or `GENINPUT` routines the values to return in `X$` when the user clicks each button. This 2-dimensional array contains control IDs and their `X$` value to return if operated. Here is a code sample from `ARACC3` that illustrates how to use the array:

```
18500 REM 18500,5 "----- Setup Button Keys for HLE's
18505 GUI_BUTTONS:
18510 IF GUI=0 THEN RETURN
18515 DIM BUTTON_KEYS$(1:6,1:2); LET BUTTON_KEYS$(1,1)="9000",
18515:BUTTON_KEYS$(1,2)=$0D$,BUTTON_KEYS$(2,1)="9010",BUTTON_KEYS$(2,2)="A",
18515:BUTTON_KEYS$(3,1)="9020",BUTTON_KEYS$(3,2)="H",BUTTON_KEYS$(4,1)="9030"
,
18515:BUTTON_KEYS$(4,2)="N",BUTTON_KEYS$(5,1)="9040",BUTTON_KEYS$(5,2)="P",
18515:BUTTON_KEYS$(6,1)="9050",BUTTON_KEYS$(6,2)="G"
18520 RETURN
```

This code sets up 6 buttons. The first series of elements holds the control ID of each button, and the second holds the value to be returned in `X$` if the user clicks on the button. Use `$0D$` to return the **Enter** key, and `$09$` to return the **Tab** key.

You can assign the values to the `BUTTON_KEYS$[ALL]` array anywhere within the program (before you `CALL GENLINE` or `GENINPUT`). However, it is more convenient to add a subroutine as in the above example.

Add the `GUI_LN[ ]` Array to HLE Screens

You must tell `GENLINE` exactly where each column in the `LN$[ ]` array ends. For example, in the Inventory Price Structures HLE, the `LN$[ ]` array contains this data from sample data (the column numbers in the first two rows are not in the array, they are only for reference):

```
1111111111222222222233333333334444444444555555555566666666667777
123456789012345678901234567890123456789012345678901234567890123
```

Cust Lvl	Description	Adjustment Base	%/ \$	Adj Amount
NONE	Price Structure Default	Calc. price	\$	.0000
ACE001	Ace Builders Contract	Base cost	%	10.0000
JOBBER	Jobber Customer	Calc. price	%	-10.0000
RETAIL	Retail Customer Pricing	Calc. price	\$	.0000
WHSL	Wholesale Customer	Base cost	%	10.0000

Note how the columns are justified and the positions at which they end (include any spaces after character fields unless it followed by a numeric field):

Column Name	End Pos	Justification
Customer Level	10	Left
Description	37	Left
Adjustment Base	55	Left
Type (\$/%)	56	Center
Adjustment Amount	73	Right

Use the GUI_LN[] array to pass this information to GENGULIN.PUB, which uses it to determine how to display data in each column of the grid. This sample from the SETUP section of INACPS illustrates the use of the GUI_LN[] array:

```
1472 LET GUI_LN[1,1]=10,GUI_LN[2,1]=37,GUI_LN[3,1]=55,GUI_LN[4,1]=56,
1472:GUI_LN[5,1]=73
1474 LET
GUI_LN[1,2]=0,GUI_LN[2,2]=0,GUI_LN[3,2]=0,GUI_LN[4,2]=2,GUI_LN[5,2]=1
```

GUI_LN[] is a two-dimensional array. The first series of elements stores the ending character position for each column. The second set of elements stores what type of justification to use for each column. Use 0 for left-justified, 1 for right-justified, and 2 for centered. If the field is a YES/NO only field, you can also set the second element to 3 for a check box.

As with standard screens, alphanumeric data should be left-justified with the column header, and numeric data is right justified.

Checklist

Use the checklist below as a guide when you convert your programs to run in both character and graphical modes.

Application _____

Done	Task	Initials/Date
	Prepare programs for graphical file creation (see page 4-40)	
	Run graphical screen creation utility (see page 4-41)	
	Clean up resource file (correct original program and rerun utility if necessary) and add control ID 4999 (see page 4-42)	
	Merge new general routines (see page 4-45)	
	Convert PROMPTS (7500) section (see page 4-45)	
	Add GUI_PROMPTS (17500) (see page 4-46)	
	Convert DATA (7700) section (see page 4-47)	
	Add GUI_DATA (17700) (see page 4-47)	
	Merge and implement new GENNUM and GENPHONE (see page 4-48)	
	Merge GENCHECK, GENCHECK_RADIO, and GENLIST where necessary (see page 4-48)	
	Add GUI_ID before GENINPUT and GENLINE calls (see page 4-48)	
	Add GUI_RADIO\$ for radio button controls (see page 4-48)	
	Merge CLEAR_GLOBAL routine, implement in program (see page 4-49)	
	Convert speedos (see page 4-50)	
	Convert non-DATA screen prints (see page 4-51)	
	Implement DISABLE_FIELDS and ENABLE_FIELDS (see page 4-51)	
	Add logic to hide and show controls where necessary (see page 4-53)	
	Add OSGUI\$ to public program CALL/ENTER lists (see page 4-54)	
	Implement hide or destroy logic to functions with multiple windows (see page 4-54)	
	Add logic to assign BUTTON_KEYSS\$[] array where necessary (Inquiry, HLE, other command functions) (see page 4-55)	
	Add logic to assign GUI_LN[] array to define columns for HLE screens (see page 4-56)	

Converting to 7.0x

Use the following information to upgrade or convert an existing version 6.x application to run on both PRO/5 and BBj language environments.

Programming Changes

Several changes are required in version 6.5 application programs to allow them to run on the BBj language platform available beginning with version 7.00. While these changes are not necessary to run these programs under the PRO/5 platform and version 7.0, it is easier to make the changes in both platforms and maintain a single set of programs that can run on either platform.

Reserved Word Checking

BBj is sensitive to the use of key words as variables in programs. Therefore, all programs must be checked for reserved words using the `_keyword` utility. Any variables identified as reserved word conflicts must then be changed.

INFO() Variable

The `INFO( )` variable contains several new values and changes to some others. Search programs for use of the `INFO( )` variable, then consult the BBj documentation and make the appropriate changes in affected programs.

Sort File Allocation

Unlike in prior versions, sort files in version 7.0 are not named using the terminal ID. This change is required because the same terminal ID can be assigned to several User IDs in version 7.0. In addition, multiple functions can be launched from the MDI Menu, which could cause conflicts in sort file naming.

To name a sort file, use `STBL("SORTSEQ")` in place of the terminal ID. Sort files should be named consistently as `"SCRT."+STBL("SORTSEQ")`, and, if necessary, `"SCRT2."+STBL("SORTSEQ")`, and so on.

Invalid Verbs

Some verbs, functions, and variables that were infrequently used in PRO/5 are no longer available in BBj. They include:

- CPL
- LST
- PGM
- DSZ
- PSZ
- SETTIME

Check programs for use of these verbs, functions, and variables and change the code as needed.

System File Changes

Enhancements to OSAS version 7.0 included changes to certain system files. Programs must be changed to accommodate these enhancements.

OSINFO.xxx

The OSINFO.xxx files changed to a text format for version 7.0 to make them easier to edit from outside OSAS. Because the information in the file is harder to obtain than from a BBx file format, the main directory paths are now stored in global variables (see the section on global variables in chapter 2 for a list).

If other information in the OSINFO.xxx file is needed in a program, you can retrieve the value for any line in the file by calling the RMPRTXT.PUB public program.

.ARC Files

BBj stores graphical form layouts in ASCII (text) resource files with a .ARC extension. PRO/5 uses binary formatted resource files with a .BRC extension, just as in OSAS version 6.x. You must convert the .BRC files to .ARC files using the BASIS utilities and include the resulting .ARC files on the BBj media.

Graphical User Interface Changes

OSAS version 7.0 makes use of a new control type (INPUTD) for date fields. There are also other changes in BBj related to the graphical user interface.

Default Fonts in BBj and Visual PRO/5

Graphical forms and controls in the Visual PRO/5 implementation of OSAS 7.0 use the MS Sans Serif font, just as they did in previous versions of OSAS.

Because the MS Sans Serif font is not reliably reproduced on all graphical operating systems under Java, all forms and controls in version 7.0 for BBj implementations use the Arial font instead. Change the default font in the .ARC files to Arial for all controls that currently use MS Sans Serif.

Images

Images on graphical forms and controls in prior versions of OSAS running on the Visual PRO/5 language platform generally used .BMP formatted graphics.

Graphical images under Java must use either the .GIF or .JPG formats to render properly. Include .GIF or .JPG versions of any graphical images on any BBj media and make the necessary changes to the programs that use them to open the correct files.

INPUTD Controls

Date fields in the Visual PRO/5 implementations of OSAS 7.0 continue to use INPUTE controls as they did in prior versions of OSAS.

For BBj implementations, version 7.0 of OSAS makes use of the new INPUTD control offered by BASIS under BBj. The new control offers several advantages.

To implement the INPUTD control in your programs, change the necessary .ARC files to use INPUTD controls instead of the INPUTE control used in prior versions. In addition, each INPUTD control should be accompanied by an associated tool button (like Inquiry fields) that allows the user to open a calendar from which they can select the date instead of entering it.

Property	Value
Control ID	<4000, increasing by 10> ¹
Name	txt<field name> (for example, txtEndDate>
X Position	<position> ²
Y Position	<position> ²
Width	<field length> ²
Height	18
Fore Color	<System Default>
Back Color	<System Default>
Font	Arial, Regular, 8
Disabled	checked ³
Client Edge	checked
Mask	leave blank
Max Length	10
Scan with +/-	checked

¹Standard OSAS screens have INPUTD controls numbered in increments of 10. This allows you room to add other controls between standard controls if necessary.

²To calculate X position, Y position, and width, refer to “Calculation of Field Sizes” on page 4-37.

³Refer to Disabling Fields later in this section.

For the associated tool button:

Property	Value
Control ID	<control ID of the input control + 11000> (for example, 15010).
Name	btnInq<input field name> (for example, btnInqEndDate)
X Position	<x-position of input control + width of the input control -3>
Y Position	<same as y-position of INPUTE/INPUTN control>
Width	16
Height	18
Fore Color	<System Default>
Back Color	<System Default>
Font	Arial, Regular, 8
Disabled	checked ¹
Client Edge	checked
Face Type	Bitmap
Bitmap File	OSDATE2.GIF

¹Refer to Disabling Fields later in this section.

Creating BBj Media

Installable media for the BBj language platform requires some special considerations.

Tokenize Programs for BBj

Although BBj can load and run programs tokenized for use with PRO/5, OSAS 7.0 for BBj implementations includes programs re-tokenized for BBj to improve performance in loading these programs for use. Convert PRO/5 tokenized programs to BBj tokenized programs using the BBjCPL.EXE utility.

Include .ARC Files

Include only the .ARC resource files in the GUI directory on the media.

Create New xxAPPL.TXT File

The installation programs for version 7.0 use a new text-format xxAPPL.TXT file that contains all of the information formerly stored in several text files on the media, including xxFRST.TXT, xxLAST.TXT, xxAPPL.TXT, and xxLABEL.TXT.

See the section on creating installable media in chapter 2 for more information about creating the new xxAPPL.TXT file.

Converting to 7.5x

Use the following information to upgrade or convert an existing version 7.x application to run under version 7.5x.

Programming Changes

Several changes may be necessary in order for 7.0 application programs to allow them to run properly under 7.5x.

Merge new GEN routines

The GENeral routines for GENINPUT, GENLINE, and GENOPEN have changed in version 7.5. These were changed to support drill down features for General Ledger, and for encrypted files across the software. Use the _update utility with your interpreter, along with SDLIB.75 to merge in the proper routines.

Drill Down Changes

In prior version of OSAS, program modification had to be done for GENINPUT.PUB to recognize the **Shift-F3** key. This was generally done by trapping the letter “B” in Y\$. With OSAS 7.5, GENINPUT now traps this key and always will check it in the GENINPUT and GENLINE merge section of code. If you are already trapping this key, you’ll need to examine how you’re reacting to this keypress, and change your code accordingly. Standard sections of OSAS will call DRILL: and DRILL_HLE: if the user hits **Shift-F3**.

Encrypted File Changes

Programs that are accessing certain files in OSAS may need to change so that the files are opened properly. By setting GENOPEN.CRYP=1 before calling GENOPEN, a program is able to tell GENOPEN if the data file is encrypted or not.

System File Changes

Enhancements to OSAS version 7.5 included changes to certain system files. Programs must be changed to accommodate these enhancements.

xxDATA.yyy

The IOLIST for XXDATA changed by adding an encryption flag to the end of the data record. This tells general programs that handle data file opening and creation whether a specific data file is encrypted or not. Make sure that you use the Data File Information function in the Software Development Utilities to make the proper changes to your xxDATA.yyy files. You can also convert your xxDATA.yyy files by using the ‘Convert 7.0x xxDATA.yyy File to 7.5x’ function within the Software Development Utilities.

xxMN

The IOLIST for XXMN changed by adding a path field to the file. This enables the menu programs to launch ODBC and other documents from the menu files. To convert your menus, use the ‘Convert 7.0x xxMN File to 7.5x’ function within the Software Development Utilities.

Graphical File Changes

.ARC Files

Two new event flags are being trapped in the BBj version of OSAS. These events are the mouse scroll wheel event, and the window move event. If you want your users to be able to use the mouse wheel to navigate thru a graphical HLE area, the event for the mouse scroll wheel must be checked. Similarly, if you want your users to be able to have overlapping windows launched from a single function move in unison with one another, you must check the window move event. These events are not available under ResBuilder. You must use the BASIS IDE, or manually fix the event flag within the resource file with a text editor.

CHAPTER 5

5

The Specification	5-1
Sample Specification	5-9

Technical Documentation

The Specification

The functional specification provides all functional and technical information for application development or enhancement. It must provide the information that a writer needs to produce the user's guide and marketing materials, that programmers need to write or modify the code, and that QA needs to test the final product against the specification.

When all members of a project team (programmers, technical writers, quality assurance analysts, client support reps, and marketing and sales people) adhere to the functional specification, they are better able to develop, document, test, support, market and sell a solid system.

When you develop a new application, the functional specification serves as the basis for the technical documentation in the Developer Tool Kit, including the *File Descriptions* manual. When you enhance an application, the functional specification serves the same purpose, but it contains only the items that change.

Each area of the specification is defined below. Samples from the Inventory 5.10 Technical Specification are provided in the next section.

Sample Specification Table of Contents

I.	General Information	
1.	Table of Contents	1-1
2.	New Product Features	1-x
3.	Data Dictionary Definitions	1-x
4.	File Relationships	1-x
II.	Functional Specification	
1.	Menu 1	2-x
2.	Function 1-1	2-x
3.	Function 1-2	2-x
4.	Menu 2	2-x
5.	Function 2-1	2-x
6.	Function 2-2	2-x
III.	File Descriptions	
1.	File List	3-1
2.	File Layouts	3-x
3.	Field Cross-Reference	3-x
IV.	Appendixes	
1.	File Conversions	A-1
2.	Tables	B-1
3.	Options and Interfaces	C-1
4.	Functional Reference Materials and Examples	D-1
5.	Subfunctional Specifications	E-1

Each of these sections is described below.

General Information

Menu/Table of Contents

The table of contents serves a dual purpose. In addition to identifying the location of information in the text, it also serves as a sample of the product menu.

New Features

List the new features of the design. In new product development the feature list may be quite long, but in enhancement development the list is much shorter. This section addresses the needs of marketing and sales, providing a nontechnical overview of the development effort.

Data Dictionary Definitions

Much of the traditional specification revolves around redundant presentation of related field information. The Data Dictionary Definitions section provides a detailed description of key fields in one location to minimize redundant references throughout the functional specification.

File Relationships

The key to successful data file design lies in understanding and clearly defining file relationships. This section contains a file relationship diagram that serves to explain these relationships.

Functional Specification

Describe in depth each menu and function in this section.

Menu

Each menu description should contain these elements:

- functional overview
- screen sample
- depiction of the Menu file entries

Maintenance/Entry Functions

Each maintenance/entry function should include these elements:

- functional description
- list of files accessed
- screen sample(s)

- program flow diagram (optional)
- field descriptions
- special logic/calls

Print Functions

Each print function should include these elements:

- functional description
- files accessed
- pick screen sample
- pick screen field descriptions
- special logic/calls
- report sample
- report field descriptions

File Descriptions

The File Descriptions section describes the contents of each data file used in the application (or, in the case of an enhancement, the files modified). If the application is based on a data dictionary, include a Field Cross-Reference section to verify the consistency of common variables (such as Item ID) between files. The Software Development Utilities contain functions that help you define, list and cross-reference your data files.

File List

The File List describes each application data file, including general information about the file's usage, such as FILE[] array index, RW topic number, and so on.

File Layouts

The File Layouts include specific information about each file: IOLIST or string template definition used to access the file's data, dimensions or fields, field descriptions, record size, field types and sizes, and a general description of the file. See the sample that follows. Each file layout begins with the file name and the purpose of the file.

File Definition Section

Channel Index	The index to the FILES[ALL] array generally used for the file
IOLIST or Template	The IOLIST or sting template used to access the file
Array Sizes	A list of the dimension sizes for arrays used in the file
String Sizes	A list of the dimension sizes for strings used in the file
RW Topic No.	The Report Writer topic number for the file (if any)
Key Definition	A list of the key definitions for an Mkeyed file (see sample below)

Main Record Layout Header

File Name	The name of the file
File Type	Mkeyed, Direct or Indexed
Record Size	The number of bytes each record requires and the actual record size of the file
# of Keys	The number of keys in an Mkeyed file

Main Record Layout

Field Number	The number assigned to each physical field in the file Multiple logical fields, which may make up a physical field
Field Name	The variable used in the application when referring to this logical field
Description	The description of the field; you can use multiple lines for the description if necessary
Format	For an alphanumeric variable, A followed by the length of the string for a numeric variable, N followed by the maximum length of the value, including the decimal point and the negative sign

Notes	Additional field information. For string fields, the notes may include valid entries or a mask for strings containing numbers (for example, sequence numbers). For numeric variables, the note must be <i>Julian</i> for dates, <i>Boolean</i> for boolean variables, the user-defined mask (i.e., <i>DOLL</i> or <i>IQTY</i>), or a depiction of the numeric mask in the format <i>x.y</i> (where <i>x</i> is the number of characters to the left of the decimal and <i>y</i> is the number of characters to the right of the decimal). If the variable can be negative, the negative sign must be displayed.

Field Cross-References

The Field Cross-Reference List shows the field names used in a data dictionary file definition sorted by field name and then by file name. This list helps you verify that you have named common fields consistently between files, which is necessary to provide ODBC/SQL links between data files.

Appendixes

The appendixes to the technical specification contain additional information about extra- or cross-functional programs, files and concepts. Extra-functional topics may include help, inquiry window definitions, Global Inquiry window definitions, EIS functions, options and interfaces, tables, file conversions and so on. Cross-functional topics include concepts that may involve a number of functions (for example, pricing in Inventory 5.1, which involves several maintenance and entry programs).

File Conversion

This section lists each field in each file to be converted and the field from the old file layout that will be used as a basis for the new field.

Tables

This section lists the contents of the tables used in the system.

Options and Interfaces

This section lists the records for the xxOI file and the positions of the resulting options in the options string.

Functional References and Examples

The contents of this section depend on the application for which you are writing the specification. It may include special information for documentation or quality assurance, general information about a complex concept, or examples of system setup. In the Inventory 5.10 Specification, this section included a grid demonstrating the effect of various transaction types on the inventory quantity buckets, and examples of price calculations using the new pricing scheme.

Subfunctions

This section contains a function specification of special subfunctional programs in the application. For example, the Inventory 5.10 Specification uses this section to describe the Inventory Price Calculator and the Item Lookup subfunctions.



Sample Specification

A sample technical specification begins on the next page.

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****TABLE OF CONTENTS -- SAMPLE SPECIFICATION****Chapter 1: General Information**

- Table of Contents (1-1)
- New Features (1-2)
- File Relationships (1-3)
- Data Dictionary Definitions (1-4)

Chapter 2: Functional Specification

- File Maintenance (2-1)
 - Item Maintenance (2-2)
- MasterFile Lists (2-8)
- Item Detail List (2-9)

Chapter 3: File Layouts

- File List (3-1)
- File Layouts (3-2)

Appendixes

- File Conversion (A-1)
- Tables (B-1)
- Options and Interfaces (C-1)
- Inquiry Window Definitions (D-1)
- Pricing Examples (E-1)
- Transaction and Quantities Table (F-1)
- Item Lookup Routine (G-1)

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

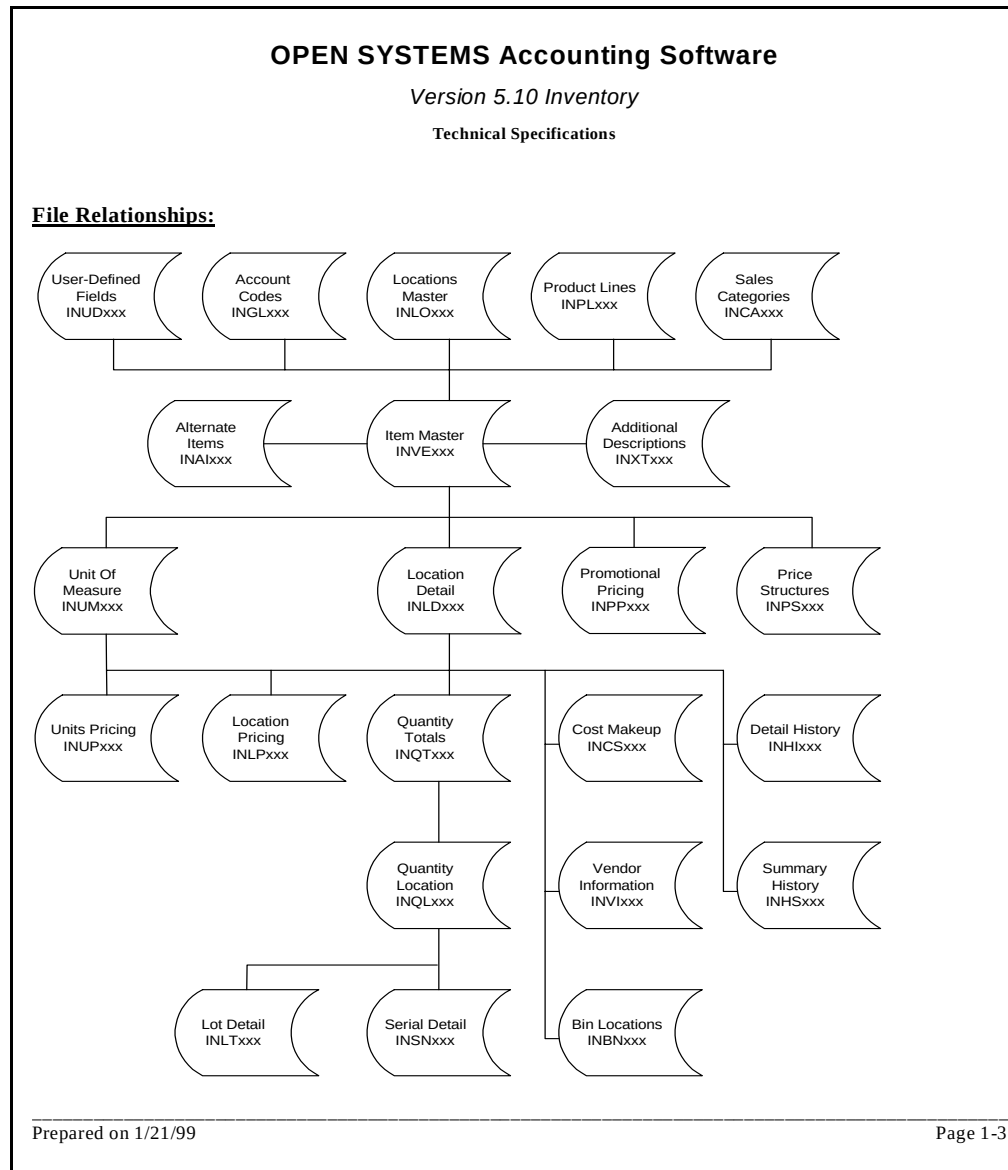
Features of Inventory 5.10

New Item Information available for each inventory item

- Warehouse and item information has been separated. Defaults are established at the item level, with overrides by warehouse location.
- Ability to identify many different alternate items and search for any part number. Alternate items may be used for customer's part number, vendor's part number, UPC codes, and alternate or superseded items.
- User-defined fields may be entered, defining up to four additional sorts. Many Inventory reports and processes can be selected using these fields. Up to 16 user-defined fields may be entered.
- Reorder information may be entered for each item. System-generated reorder information includes order point, safety stock and EOQ. Both Minimum and Maximum Order fields can be entered.
- Unlimited movement history is retained for 10 transaction types: sales and credits, purchases and debits, builds and issues, material requisitions, transfers in and out, and adjustments.
- Pricing may be done through a matrix of customer level and price ID, item-specific pricing with quantity breaks, and promotional pricing.
- Average, base, list and minimum prices are set for each unit of measure for an item.
- Standard costs with a cost makeup have been implemented. Purchase price variance will be posted when goods are received at a cost other than standard cost.
- Actual costs will be stored for unlimited levels of LIFO or FIFO tiers based on receipt date, even if average or standard costing methods are used.
- Vendor history, lead time, and quantity break pricing is stored for an unlimited number of vendors for each item.
- Multiple bins are maintained for each item for physical count processing. Last physical count information is maintained.
- Full lot tacking is maintained with expiration dates, vendor history, sales history and transaction history.

New Codes define new functionality

- Category has been expanded into three fields: Product Line for inventory reporting, Sales Category for sales analysis reporting and GL Account Codes for account definition.
- User-defined fields may be validated by entering valid items in the user-defined categories file.
- Forecast types allow unlimited usage of forecast calculations for reordering. Seasonal, trend, and horizontal forecasting are available.
- GL account codes identify GL postings for inventory, sales and cost of goods sold. The inventory adjustments account separates adjusting type entries from the inventory account for COGS adjustments, purchase price variances, physical count variances and transfer cost adjustments.
- Customer levels define the customer side of the new pricing matrix; price ID defines the inventory side. Combine them for flexible pricing in the Price Structures function.
- Define special promotional pricing by entering a range of product lines, items, user-defined categories, customer levels or price IDs. Once the selection criterion has been established, adjust prices from base price or from cost, using a percentage or fixed amount.



OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications**

Element Name	Pseudonyms	Format	Len	Definition
Item ID	Item Number Part Number	a/n	20	Key to Item files
Location ID	Warehouse ID	a/n	6	Key to Location files
Lot Number		a/n	16	Key to Lot files
Serial Number		a/n	35	Key to Serial files
Units	UOM	a/n	6	Unit of measure code Key to Units files
Initial Date	Date	mm/dd/yyyy	7	Generally date received Key to Quantity Locations file
Price ID	Price Structure	a/n	12	Item classification for price matrix calculations. Defines like items together for pricing. Key to Price Structure file.
Customer Level	Price Level	a/n	12	Customer classification for price matrix calculations. Defines like customers together for pricing. Second key to Price Structure file.
Base Units		a/n	6	The unit of measure that all quantities are stored in. Generally, the base unit should be the smallest unit of measure.
Default Units	Units Stocking Units	a/n	6	The unit of measure all screens and reports will reflect unless otherwise specified. This should be the stocking unit of measure.
Units Conversion Penalty flag	/%\$	flag	1	Flag that determines how to calculate conversion penalties. (See Conversion Penalty) \$=Add \$ amount to price %=Add % to price
Product Code	Product Line Product Class Product Category	a/n	12	A code used to group like items together for inventory reporting.
User-Definable Categories		a/n	12	A series of codes the user may use to define report sequences and selections.
User-Definable Fields		a/n	12	A series of codes the user may use to define report selection criteria. Comment fields.
Tax Class		n - 0 to 9	1	A code to group items for tax calculations.
Sales Category	Category	a/n	2	A code used to group items together for sales analysis reporting.
Alternate Items	Vendor's Part Number	a/n	20	A series of alternate item codes that allow accessing inventory by something other than the item ID.

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

File Maintenance

Functional Description:

Use the functions on the FileMaintenance menu to set up and maintain items, locations for items, locations, price structures, promotional pricing, and tables. The Tables function is the standard RMACT program and is not described here.

When you select *File Maintenance* from the Inventory Main Menu, this screen appears:

```
00000000001111111111222222222233333333334444444444555555555566666666667777777777
0123456789012345678901234567890123456789012345678901234567890123456789
+-----+
mm/dd/yyyy          Company a - aaaaaaaaaaaaaaaaaa          99:99 aM
+-----+
      +- Main Menu -----+
      |                     |
      | +- Inventory -----+
      | |                     |
      | | +- File Maintenance ----+
      | | |                     |
      | | | Items                |
      | | | Item Locations       |
      | | | Locations            |
      | | | Price Structures     |
      | | | Promotional Pricing  |
      | | | Tables               |
      | | +-----+
      | +-----+
+-----+
Taa----- OPEN SYSTEMS (R) Accounting Software -----+
24
```

Menu ID: **FMAINT** Title: **File Maintenance**

Line	Description	Type	Program	Parameter
01	Items	3	INACI	
02	Item Locations	3	INACL	
03	Locations	3	INACLO	
04	Price Structures	3	INACPS	
05	Promotional Pricing	3	INACPP	
06	Tables	3	RMACT	F

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

Items Selection Screen

Program: INACI

Data files required:

INVE - Item Masterfile (FILE[2])

Key=Item

000000000011111111222222223333333344444444555555556666666677777777

012345678901234567890123456789012345678901234567890123456789

----- Items -----

Item ID

aaaaaaaaaaaaaaaaaaaa

Copy From

aaaaaaaaaaaaaaaaaaaa

Description

aaaaaaaaaaaaaaaaaaaa

Do You Want to Maintain:

General Information?

Units of Measure?

Alternate Items?

Item Location Information?

TTT

TTT

TTT

TTT

Functional Description:

The Items selection screen appears when you select the Items function from the menu. Use it to enter an item ID that you want to add, or to enter or select an item ID that you want to maintain or delete. Depending upon the selections, the Items selection program will chain to a variety of other programs as listed.

Prepared on 1/21/99

Page 2-3

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****INACI fields:**

Field Name	Variable Name	Len	Mask	Inquiry	Maint	Valid/Verify/Formulas/Notes
Item ID	INVE.ITEM\$	20		RMITEM		Use RMITEM.PUB
Copy From	COPY\$	20		INITEMW1		Only if Item ID Not Found *See <i>Copy From</i> below
Description	INVE.DESC\$	35				
Additional Description lines	INXT.DESC\$	35 ea.				*See <i>Additional Descriptions</i> below
General Information?	SELECT [1]	3				Valid entries: Y=Yes (Default) N=No
Units of Measure?	SELECT [2]	3				Valid entries: Y=Yes (Default for add) N=No (Default for edit)
Alternate Items?	SELECT [3]	3				Valid entries: Y=Yes (Default for add) N=No (Default for edit)
Item Location Information?	SELECT [4]	3				Valid entries: Y=Yes (Default for add) N=No (Default for edit)

Special logic/calls: None

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

Copy From

If you enter a valid ID in the Copy From field, you'll see this copy window:

```
00000000001111111111222222222233333333334444444444555555555566666666667777777777
01234567890123456789012345678901234567890123456789012345678901234567890123456789
----- Items -----
Item ID      +----- Copy From -----+aa
Description  Copy From      aaaaaaaaaaaaaaaaaa
              Description  aaaaaaaaaaaaaaaaaa
Do You Want  Do You Want to Copy:
Gene         Alternate Items?          TTT
Unit         User-Defined Fields?       TTT
Alte         Location Information?      TTT
Item         Price Information?         TTT
              Customer Level Pricing?  TTT
              Cost Summary Information? TTT
              Vendor Information?       TTT
              Bin Information?          TTT
```

Field Name	Variable Name	Len	Mask	Inquiry	Maint	Valid/Verify/Formulas/Notes
Alternate Items?	COPY [1]	3				1) Valid entries: Y=Yes or N=No
User-Defined Fields?	COPY [2]	3				1) Valid entries: Y=Yes or N=No
Location Information?	COPY [3]	3				1) Valid entries: Y=Yes or N=No
Price Information?	COPY [4]	3				2) Valid entries: Y=Yes or N=No
Cust Level Pricing?	COPY [5]	3				2) Valid entries: Y=Yes or N=No
Cost Summ Information?	COPY [6]	3				2) Valid entries: Y=Yes or N=No
Vendor Information?	COPY [7]	3				2) Valid entries: Y=Yes or N=No
Bin Information?	COPY [8]	3				2) Valid entries: Y=Yes or N=No

Prepared on 1/21/99

Page 2-5

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****Special logic/calls:**

- 1) Defaults to YES.
- 2) Defaults to COPY[3] value. If COPY[3]=0, change these fields to NO, with no edit.

Copy Logic:

Read INVExxx, Key=COPY\$, DOM=07=8 msg; if found, prompt for Copy From choices as shown

When the user Proceeds, change the Item ID to the new ID

For each record in INUMxxx for this item, change the item ID and write the record

If COPY[1]=1 then

For each record in INAIxxx for this item,

change the Item ID and write the record

If COPY[2]=0 then

for 1 to 16, let INVE.UDF\$[x]=FILL[12], next

Write to INVExxx and exit

If COPY[3]=0 then DIMension COPY[0] and exit else

let TOTCS1=0

for each record in INLDxxx for this item,

change the Item; zero INLD.LCST; DIM all override flags, last sale and purchase information (INLD.SADT through INLD.LEAD

If COPY[4]=1 then

for each record in INUPxxx for this item and location,

change the Item ID and write the record

If COPY[5]=1 then

for each record in INLPxxx for this item and location,

change the Item ID and write the record

If COPY[6]=1 then

for each record in INCSxxx for this item and location,

change Item ID, write the record, add INCS.CSTC to TOTCS1

If COPY[7]=1 then

for each record in INVxxx for this item and location,

change the Item, DIMension all INV.Lxxx (last PO information) and write the record

If COPY[8]=1 then

for each record in INBNxxx for this item and location,

change the Item ID and write the record

let INLD.ACST=TOTCS1; write the record to INLDxxx

Technical Specifications

Page 2-7

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

Master File Lists

Functional Description:

Use the functions on the Master File Lists menu to print a list of items, locations for items, locations, price structures, promotional pricing, and tables.

When you select *Master File Lists* from the Inventory Main Menu, this screen appears:

```
000000000011111111222222222233333333334444444444555555555566666666667777777777
0123456789012345678901234567890123456789012345678901234567890123456789
+-----+
mm/dd/yyyy      Company a - aaaaaaaaaaaaaaaaaa      99:99 aM
+-----+
+-- Main Menu -----+
+-- Inventory -----+
+-- Master File Lists -----+
Item Detail List
Item Summary List
Item Locations List
Location Detail List
Price Structure List
Promotional Pricing List
Tables List
+-----+
Taa----- OPEN SYSTEMS (R) Accounting Software -----
```

Menu ID: **MASTER** Title: **Master File Lists**

Line	Description	Type	Program	Parameter
01	Item Detail	3	INPRID	
02	Item Summary	3	INPRSUM	
03	Item Locations	3	INPRIL	
04	Locations	3	INPRLO	
05	Price Structures	3	INPRPS	
06	Promotional Pricing	3	INPRPP	
07	Tables	3	RMPRTB	

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

Item Detail File List

Functional Description:

Use the Item Detail List function to print a list of the records in the Items file, the Units of Measure file, the Alternate Items file, and the Item Location file.

Items Detail Selection Screen

Program: INPRID

Data files required: INVE - Item Masterfile (FILE[2]) Key=Item
INUM - Units of Measure (FILE[3]) Key=Item+Unit of Measure
INAI - Alternate Items File (FILE[4]) Key=Item+Type+Src ID+Alt Item
INXT - Add'l Descriptions (FILE[20]) Key=Item+Sequence Number
INLO - Locations File (FILE[9]) Key=Location ID
INLD - Location Detail File (FILE[8]) Key=Item+Location ID
INUD - User-Defined Fields (FILE[36]) Key=Field Number+Field Value

When you select *Item Detail* from the Master File Lists menu, you see the following screen:

```
00000000001111111112222222223333333333444444444555555555666666666777777777
0123456789012345678901234567890123456789012345678901234567890123456789
----- Item Detail List -----
Pick Item ID From aaaaaaaaaaaaaaaaaa Print by
Thru aaaaaaaaaaaaaaaaaa 1. Item ID
Product Line From aaaaaaaaaaaaaa 2. Product Line
Thru aaaaaaaaaaaaaa 3. User-Defined 1
User-Defined 1 From aaaaaaaaaaaaaa 4. User-Defined 2
Thru aaaaaaaaaaaaaa 5. User-Sort 1
User-Defined 2 From aaaaaaaaaaaaaa 6. User-Sort 2
Thru aaaaaaaaaaaaaa 9
Print General Information? TTT
Additional Descriptions? TTT
Units of Measure? TTT
Alternate Items? TTT
Location Summary? TTT
```

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications**

Functional Description: The Item Detail List screen appears when you select the Item Detail List function from the menu. Use it to enter a range of item IDs, product lines, and user-defined fields that you want to print a list for of the information stored. Depending upon the selections, the Item Detail List program will chain to a variety of other programs as listed.

INPRID fields:

Field Name	Variable Name	Len	Mas k	Inquiry	Maint	Valid/Verify/Formulas/Notes
Item ID From	IFROM\$	20		INITEMW1		
Item ID Thru	ITHRU\$	20		INITEMW1		
Product Line From	PFROM\$	2		INPRODR1		
Product Line Thru	PTHRU\$	2		INPRODR1		
User-Defined 1 From	UDFROM\$ [1]	2		INUSRDR1 SELVAL\$= "01"+FILL(12)		Prompt "User-Defined 1" will be replaced from INUSRxxx table.
User-Defined 1 Thru	UDTHRU\$ [1]	2		INUSRDR1 SELVAL\$= "01"+FILL(12)		
User-Defined 2 From	UDFROM\$ [2]	2		INUSRDR1 SELVAL\$= "02"+FILL(12)		Prompt "User-Defined 2" will be replaced from INUSRxxx table.
User-Defined 2 Thru	UDTHRU\$ [2]	2		INUSRDR1 SELVAL\$= "02"+FILL(12)		
Additional Descriptions?	SELECT\$ [1]	3				2) Valid entries: Y=Yes or N=No
General Information?	SELECT\$ [2]	3				1) Valid entries: Y=Yes or N=No
Units of Measure?	SELECT\$ [3]	3				2) Valid entries: Y=Yes or N=No
Alternate Items?	SELECT\$ [4]	3				2) Valid entries: Y=Yes or N=No
Item Location Information?	SELECT\$ [5]	3				2) Valid entries: Y=Yes or N=No
Print By	PB	1				Prompt "User-Defined 1" and "User-Defined 2" will be replaced from INUSRxxx table.

1) Defaults to YES

2) Defaults to No

Technical Specifications

Item Detail List (INPRID1)

Page 9999

Page 2-12

Technical Specifications

Page 2-13

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****INPRID1 fields:**

Field Name	Variable Name	Length	Mask	Valid/Verify/Formulas/Notes
Item ID	INVE.ITEM\$	20		Print on every page heading.
Description	INVE.DESC\$	35		Print on every page heading.
Additional Description	INXT.ADDL\$	35		Key=INVE.ITEM\$+Sequence Number From 01 thru 10
Status	INVE.STAT\$	12		A=Active (Default) D=Discontinued S=Superseded O=Obsolete
Item Type	INVE.ITYP\$	1		1=Serialized 2=Nonserialized 3=Service
Kitted Item?	INVE.KITS	3		1=Yes 0=No
Track Lots?	INVE.LOTS	3		1=Yes 0=No
Auto Reorder?	INVE.REORD	3		1=Yes 0=No
Base UOM	INVE.BUOM\$	5		
Sales Category	INVE.SCAT\$	2		
Product Line	INVE.PROD\$	12		
Item Price ID	INVE.PRID\$	6		
Tax Class	INVE.TAXC\$	2		Description for Tax Class is stored in RMCDxxx. Key is "TAXCLS-----"+INVE.TAXC\$.
Item Weight	INVE.WT	14	IQTY	
(User-Defined Field xx prompt from INUSRxxx table). xx is 01-16.	INVE.UDxx	12		
(User-Defined Field Description xx)	INUD.DESC\$			Descriptions for the User-Defined fields is stored in the INUD file. Key is Field Number+Field Value

Special logic/calls:

Sequences 01 and 02 are written as a unique key in the INVExxx file.

Two keys are written with combinations of these fields as per definition in INUDSxxx table.

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****Inventory File List**

Sec.	File Name	Description	Old File	FILE[]	RW#
1	INAIxxx	Alternate Items		4	
2	INBNxxx	Bin Locations		5	
3	INCAxxx	Sales Categories	INCAx	31	
4	INCBxxx	Physical Count Batches		21	
5	INCCxxx	Cost Makeup Codes		37	
6	INCLxxx	COGS Journal	INCDx	26	
7	INCLxxx	Customer Levels	INCDx	32	
8	INCSxxx	Cost Makeup		7	
9	INCTxxx	Physical Counts	INCTx	22	
10	INFTxxx	Forecast Types		33	
11	INGLxxx	Account Codes		34	
12	INHlxxx	Detail History	INHlx	23	
13	INHSxxx	Summary History		24	
14	INKYxxx	Alternate Keys		6	
15	INLDxxx	Location Detail	INVEx	8	501
16	INLHxxx	Lot History		27	
17	INLOxxx	Location Master	INWHx	9	
18	INLPxxx	Location Pricing		10	
19	INLTxxx	Lot Detail		11	
20	INPLxxx	Product Lines	INCAx	35	
21	INPPxxx	Promotional Pricing		12	
22	INPSxxx	Price Structures	INCDx	13	
23	INQLxxx	Quantity Locations	INVEx	14	
24	INQTxxx	Quantity Totals	INVEx	15	
25	INRQxxx	IN Requisitions		29	
26	INSHxxx	Serial History	INSHx	28	
27	INSNxxx	Serial Detail	INSIx	16	
28	INTB	Tables File	INTB	17	
29	INTLxxx	Transfer Lot/Serial IDs		40	
30	INTRxxx	Transactions	INTRx	25	
31	INTTxxx	Transactions--Transfers	INTRx	30	
32	INUDxxx	User-Defined Fields		36	
33	INUMxxx	Units of Measure		3	
34	INUPxxx	Units Pricing		18	
35	INVExxx	Item Master	INVEx	2	500
36	INVIxxx	Vendor Information		19	
37	INXTxxx	Additional Descriptions	INDEx	20	
38	xxLSxxx	Transaction Lot/Serial IDs	xxOSx	39	

Prepared on 1/22/99

Page 3-1

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

The Alternate Items file stores the alternate IDs for each inventory item, including aliases, vendor and customer part numbers, supersede part numbers, bar codes and quantity alternates. There is one record for each alternate ID for each item.

Channel Index: 4

Template: DIM INAI\$:"ITEM:C(20*),SRC:C(6*),RSV1:C(1*),
ALTI:C(20*),RSV2:C(1*),ATYP:C(1*),SDAT:N(7*),
EDAT:N(7*),RSV3:C(1*),ISVA:C(1*),ISVN[2]:N(1*)"

RW Topic No: 511

Key Definitions:

Key Number	Field	Description
0	[1:1:20]+ [6:1:1]+ [2:1:6]+ [4:1:20]	Item ID + Alternate Item Type + Source ID + Alternate Item ID
1	[4:1:20]+ [2:1:6]+ [1:1:20]	Alternate Item ID + Source ID + Item ID +
2	[6:1:1]+ [2:1:6]+ [4:1:20]+ [1:1:20]	Alternate Item Type + Source ID + Alternate Item ID + Item ID
3	[1:1:20]+ [6:1:1]+ [2:1:6]+ [7:1:7]+ [4:1:20]	Item ID + Alternate Item Type + Source ID + Start Date + Alternate Item ID
	----- 16 segments	

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****IN Alternate Items Record**

File Name: INAIxxx

Record Size: 79 (128)

File Type: Mkeyed - Dynamic

Number of keys: 4

Field No	Template	Description	Type	Length	Note
1	ITEM	Item ID	C	20	
2	SRC	Source ID	C	6	Customer/Vendor ID
3	RSV1	Reserved for OSD	C	1	
4	ALTI	Alternate Item ID	C	20	
5	RSV2	Reserved for OSD	C	1	
6	ATYP	Alternate Item Type	C	1	A = Alternate C = Customer Part L = Alias S = Supersede U = UPC Bar Code V = Vendor Part
7	SDAT	Start Date	N	7	Julian
8	EDAT	End Date	N	7	Julian
9	RSV3	Reserved for OSD	C	1	
10	ISVA	Reserved for ISV	C	1	
11	ISVN[1]	Reserved for ISV	N	1	
12	ISVN[2]	Reserved for ISV	N	1	

Prepared on 1/22/99

Page 3-3

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory*

Technical Specifications

Appendix A: File Conversion**Inventory Version 5.0 File Conversion**

From Version 4.5 Files	To Version 5.0 Files
INCAx	INPLxxx INCAxxx
INCDx	INCLxxx INLPxxx INPSxxx
INCTx	Not Converted
INDEx	INXTxxx
INHx	INHxxx INHSxxx
INSHx	INSHxxx
INSIx	INSNxxx
INTB	INTB
INTG	Not Converted
INTRx	Not Converted
INTXx	RMCDxxx
INVEx	INVExxx INBNxxx INQTxxx INLDxxx INQLxxx INGLxxx INUMxxx INVxxx INAIxxx INKYxxx
INWHx	INLOxxx

The following version 5.0 files are not converted:

INCBxxx	INCJxxx
INCSxxx	INCTxxx
INFTxxx	INLHxxx
INLTxxx	INPPxxx
INTRxxx	INUDxxx

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory*

Technical Specifications

Alternate Item RecordFile Name: INAIxxx
File Type: Mkeyed - DynamicRecord Size: 109 (128)
Number of Keys: 4

No.	Field	Description	Format	From File(s)/Flds
1	ITEM	Item ID	A 20	INVE _x 1 I0\$(1,20)
2	SRC	Source ID	A 6	INVE _x 3 I2\$(16,6) INVE _x 3 I2\$(42,6) INVE _x 3 I2\$(68,6)
3	RSV1	Reserved for OSD	A 2	N/A
4	ALTI	Alternate Item ID	A 20	INVE _x 2 I1\$(72,20) INVE _x 3 I2\$(22,20) INVE _x 3 I2\$(48,20) INVE _x 3 I2\$(74,20)
5	RSV2	Reserved for OSD	A 1	N/A
6	ATYP	Alternate Item Type	A 1	A For I1\$(72,20) V For the rest
7	SDAT	Start Date	N 7	-1
8	EDAT	End Date	N 7	-1
9	RSV3	Reserved for OSD	A 3	N/A
10	ISVA	Reserved for ISV	A 0	N/A
11	ISVN[0]	Reserved for ISV	N 1	N/A
12	ISVN[1]	Reserved for ISV	N 1	N/A

Prepared on 1/22/99

Page A-2

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

Appendix B: Tables

Functional Description:

The tables in use for Inventory 5.0 are used to store information that will not be stored in its own file. Examples include the prompts for user-defined fields, the user-defined sorts, the default code used for materials in standard cost rollups, and the current Inventory fiscal year. You set up tables using the Tables function on the File Maintenance menu. You can print a list of the tables set up using the Tables List function on the Master File Lists menu.

Tables -- INUDSxxx

Table ID: INUSRxxx **Description:** User-Defined Fields Prompts
Number of Columns: 2 **Column Width:** 12 **Type:** A

UD Field No.	Prompt
01	aaaaaaaaaaaa
02	aaaaaaaaaaaa
03	aaaaaaaaaaaa
04	aaaaaaaaaaaa
05	aaaaaaaaaaaa
06	aaaaaaaaaaaa
07	aaaaaaaaaaaa
08	aaaaaaaaaaaa
09	aaaaaaaaaaaa
10	aaaaaaaaaaaa
11	aaaaaaaaaaaa
12	aaaaaaaaaaaa
13	aaaaaaaaaaaa
14	aaaaaaaaaaaa
15	aaaaaaaaaaaa
16	aaaaaaaaaaaa

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory*

Technical Specifications

Tables -- INUDSORT

Table ID: INUDSORT **Description:** User-Defined Field Sorts
Number of Columns: 3 **Column Width:** 2 **Type:** A

Primary	Secondary	Tertiary
01	02	03
02	03	01

Tables -- INPDxxx

Table ID: INPDxxx **Description:** Fiscal Year Definition
Number of Columns: 3 **Column Width:** 12 **Type:** N

Unused	Fiscal Year	# Periods/Yr
0	9999	12

Tables -- INSTDxxx

Table ID: INSTDxxx **Description:** Standard Cost Materials Code Default
Number of Columns: 1 **Column Width:** 12 **Type:** A

Matrl Code
MA

Tables -- OPTxxx

See Options and Interfaces Section -- Appendix C

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****Appendix C: Options and Interfaces**

The Options and Interfaces for Inventory are as follows:

Option	Type	Group	Description	Values
01	I	1	Interface with General Ledger?	NO,YES
02	O	1	Valuation Method	FIFO,LIFO,Average,Standard
03	O	1	Allow Editing of Quantities?	NO,YES
04	O	1	Zero Quantity Sales Costing	Average, Last, Base, Standard
05	O	1	Use Additional Descriptions?	NO,YES
06	O	1	Keep Summary History?	NO,YES
07	O	1	Keep Detail History?	NO,YES
08	O	1	Keep Lot History?	NO,YES
09	O	1	Keep Serial History?	NO,YES
10	O	1	Show Cost in Sales Transactions?	NO,YES
11	O	1	Post Detail to General Ledger?	NO,YES
12	O	1	Check for Aliases on Report Options?	NO,YES
13	O	1	Transaction Entry Method?	Serial/Lot, Lot/Serial
14	O	1		
15	O	1		
16	O	1		
17	O	1		
18	O	1		
19	O	1		
20	O	1		

OPEN SYSTEMS Accounting Software

Version 5.10 Inventory

Technical Specifications

Appendix D: Inquiry Windows

Application: IN **Window ID:** INITEMW1 **Filename:** INVE_x
Key Number: 2 **Title:** Item Inquiry
Start Col: 0 **Start Row:** 4 **Incl Beg:** **Incl End:**
Total Col: 78 **Total Row:** 15 **Excl Beg:** **Excl End:**
Return Column: 1

No.	Data Name	Type	User Mask	Format	Field	Beg	Len	Col
1	Item ID	1			1	1	20	2
2	Description	1			2	1	35	23
3	Product Line	1			10	1	12	59
4	Item Status	1			4	1	1	74
5								
6								
7								
8								
9								
10								

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****Appendix E: Pricing Examples****The Adjustment Bases:**

Average Price These adjustment bases are fields in the item location function. Technically, they are stored in the Unit of Measure Pricing and the Location Detail files.
List Price
Base Price
Minimum Price
Base Cost
Standard Cost

No Base This adjustment base is not a base at all. You must choose dollars for the type,
and the adjustment amount is the price.

Calculated Price This adjustment base is actually the result of the previous calculation (if any).

The calculation:

If the adjustment type is %, the price is $(\text{adjustment base} * \text{adjustment amount}) + \text{adjustment amount}$
If the adjustment type is \$, the price is $(\text{adjustment base} + \text{adjustment amount})$

Examples of price calculations are shown on the next page.

OPEN SYSTEMS Accounting Software						
Version 5.10 Inventory						
Technical Specifications						
Exhibit A		Date October 31, 1995		----Customer Levels-----		
Item 100, Location A, Price ID HDWE				RETAIL	WHSL	
				5 Each	10 Case	
INUMx,INUPx	UOM	EachBase	100.00	100.00	960.00	
		10	98.00			
		100	96.00			
		1000	92.00			
	Case	Base	980.00			
		10	960.00			
		100	920.00			
INLPx	Cust Level Item 100, Location A			98.50	N/A	
INPSx	Price Structure HDWE (Base - 1%)			N/A	950.40	
INPPx	Promo: OCT (Base - 5%) MIN(Price,Promo)			N/A	902.88	
				-----	-----	
				98.50	902.88	
				x 5	x 10	
				-----	-----	
				492.50	9028.80	
Exhibit B		Date: November 1, 1995		----Customer Levels-----		
Item 100, Location A, Price ID HDWE				RETAIL	WHSL	
				5 Each	10 Case	
INUMx,INUPx	UOM	EachBase	100.00	100.00	960.00	
		10	98.00			
		100	96.00			
		1000	92.00			
	Case	Base	980.00			
		10	960.00			
		100	920.00			
INLPx	Cust Level Item 100, Location A			98.50	N/A	
INPSx	Price Structure HDWE (Base - 1%)			N/A	950.40	
INPPx	Promo: NOV 955.00/Case (No Base)					
	MIN(Price,Promo)			N/A	950.40	
				-----	-----	
				98.50	950.40	
				x 5	x 10	
				-----	-----	
				492.50	9504.00	

Prepared on 1/22/99				Page E-2		

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****Appendix F: Transactions and Quantities Table**

Transaction/Status	Quantity Committed	Quantity In Use	Quantity On Hand	Quantity On Order	Quantity Available
ADJUSTMENTS:					
Increase			Increase		Increase
Decrease			Decrease		Decrease
SALES:					
New Order	Increase				Decrease
Verify Order	Decrease	Increase			
Invoice		Increase			Decrease
Misc. Credit		Decrease			Increase
PURCHASES:					
New Order				Increase	
Goods Received			Increase	Decrease	Increase
Invoice			Increase		Increase
Misc. Debit			Decrease		Decrease

Quantity Available = Quantity On Hand - Quantity In Use - Quantity Committed

Technical Specifications

Page G -1

OPEN SYSTEMS Accounting Software*Version 5.10 Inventory***Technical Specifications****INACIFM.PUB fields:**

Field Name	Variable Name	Len	Mask	Inquiry	Maint	Valid/Verify/Formulas/Notes
Location ID	INLD.LOCA\$	6		INLOCAW1		
Item ID	INVE.ITEM\$	20		INITEMW1		
Status	INVE.STAT\$	12				Valid entries: A=Active D=Discontinued S=Superseded O=Obsolete
Item Type	INVE.ITYP\$	1				Valid entries: 1=Nonserialized 2=Serialized 3=Service
On Hand	INQT.OHQ	14	IQTY			
In Use	INQT.IUQ	14	IQTY			
Committed	INQT.CMTQ	14	IQTY			
Available	calculated	14	IQTY			AVAILABLE=INQT.OHQ- INQT.IUQ-INQT.CMTQ
On Order	INQT.OOQ	14	IQTY			
MTD Sales Qty	calculated	14	IQTY			MTD_QTY=INHS.QSAL- INHS.QSRT; 1)
MTD Sales Amount	calculated	14	DOLL			MTD_SALES=INHS.PSAL- INHS.PSRT; 1)
YTD Sales Qty	calculated	14	IQTY			YTD_QTY=INHS.QSAL- INHS.QSRT; 2)
YTD Sales Amount	calculated	14	DOLL			YTD_SALES=INHS.PSAL- INHS.PSRT; 2)
Last Sale Date	INLD.SADT	7 julian	Date			
Units	INUM.UNIT\$	5				
Base Price	INUP.BSPR	14	INUP			
Conv Factor	INUM.CONV	14	IQTY			
Penalty	INUM.PENA	14	INUP 4			
Type	INUM.PENT	1	%/\$			
Base		3	Yes			Print "Yes" for UOM when INUM.UNIT\$=INVE.BUOM\$

Special logic/calls: 1) Read INHS w/ key of "P"+Current Year+Current Period+Item ID+Location ID
 2) Read INHS with key of "Y"+Current Year+"00"+Item ID+Location ID

Prepared on 1/22/99

Page G-2

CHAPTER 6

6

Formatting	6-2
Screens and Reports	6-2
Capitalization and Style Conventions	6-3

Documentation

Documentation Standards

Documentation is an essential component of OSAS, providing assistance to and serving as a reference for users. This documentation includes user guides, manual supplements, and help text. To generate strong user documentation, follow the OSAS standards.

OSAS user guides are written in Adobe FrameMaker, and the graphics they contain are captured with Jasc Paint Shop Pro. To achieve the same look for your documentation, model your documentation's formatting on current OSAS user manuals.

Organization

The OPEN SYSTEMS Accounting Software user's manuals are reference manuals. They are divided into four parts:

- “Introduction” introduces the user to OSAS, the application, and OSAS and manual conventions.
- “Setting Up” tells the user how to prepare for using and maintaining the system.
- “Functional Reference” explains the features and uses of all the screens, windows, and processes in the application.
- “References” contains appendixes that cover such topics as conversion, system messages, file descriptions, function keys, other commands.

Formatting

If you don't have FrameMaker 6.0 and the fonts that Open Systems' writers use, use fonts that resemble those in the OSAS manuals.

Paragraphs

For regular text in paragraphs, use a 10-point serif font such as Times Roman. Indent paragraphs one inch more than subheads 1 and 2.

Subheads

Subheads are distinguished from each other by their size and their placement. Use the same sans serif font (such as Arial) for all subheads, but subhead 1 should be 14 points, subhead 2 should be 12 points, and subhead 3 should be 10 points. Place subhead 1 flush left, indent subhead 2 half an inch, and align subhead three with the text following it. Use no more than three levels of subheads; if more are needed, look for a different way to organize the text.

Chapter and Section Heads

Use the same sans serif fonts for chapter and section heads as the subheads. Chapter heads should be 24 points. Section heads should be 18 points. Chapter and section heads must always start on a right-hand page.

Screens and Reports

The quality of the screens and reports in the manual depend a great deal on the software you use to capture them. More important than the quality of the screens and reports is adherence to the standards listed in chapter 2.

Use the sample data to fill in each screen and to produce reports that are complete and valid.

Capitalization and Style Conventions

OSAS manuals use boldface, sans serif font to emphasize function and field names, buttons, keys, or user input, as well as capitalization for specific names, when directly referred to in the text. However, if you make indirect reference to a function or field name, it does not need such treatment. Generic description nouns (such as “screen,” “system,” or “menu”) are never capitalized.

Specific Names and Generic Items

Capitalize the initial letter of a specific name. When you pair a specific name with its generic description, lowercase the generic word.

Files	Transaction file
Functions	the Year-End Maintenance function
Menus	the Accounts Payable Daily Work menu
Screens	Promotional Pricing screen
Systems	Accounts Payable system
Tables	INPDxxx table

Screen and Keyboard Items

Type words exactly as they appear on the screen. Type messages in boldface sans serif font. Type terminal keys in boldface sans serif font with initial caps.

Fields	the Item Number field (or “enter the Item Number ”)
Messages	‘D’ for Delete.
Terminal Keys	Press F1 to access help.

When you make an indirect reference to a field name, do not capitalize it. Follow these examples:

Enter the customer number in the **Customer No** field.
If you entered a new customer number, enter the customer's **Name**.

Cross-References

Cross-reference by page number unless the cross-reference is to an appendix. Follow these examples:

See page 4-10 for more information.

See appendix F.

Data Entry Items

Type input items in boldface sans serif font unless the instructions for that item state otherwise. Do not type punctuation following an input item in boldface unless it is to be entered as well.

Lists

When you use bulleted lists, use round bullets. Line up the bullets with the left margin of the preceding text. Single-space lists that consist of one-line items; double-space those that consist of multiple-line items.

Notes and Cautions

Treat notes and cautions the same as the text that precedes them. If you cannot use the graphic conventions for notes and cautions, type *Note* and *Caution* in boldface capital sans serif font, and put a colon after them.

Page Numbering

Pages are numbered by chapter—1-1, 2-1, 3-1, 4-1, A-1, IX-1.

Table of Contents

The table of contents is titled *Contents*. List the chapter and section titles.

CHAPTER 7

7

Testing Phase	7-3
Design Phase	7-5
Acceptance Testing	7-13
QA Terminology	7-17

Quality Assurance

Overview

The purpose of the quality assurance process is to ensure a high-quality product. The process is generally divided into three phases: design, testing and acceptance.

Quality Assurance (QA) becomes involved in a project during the development design phase. QA participation in the design phase provides product development with support in identifying areas where product standards are not being followed, problems with data flow or follow-through to interfacing applications exist, and where the application is not user friendly. QA also helps ensure that development specifications contain information needed to draw up accurate and thorough test plans. While the software is being developed, QA prepares test strategies and writes the test plan, which includes the creation of test cases and checklists.

The testing phase begins when Software Development finishes coding and testing the product. Testing consists of system testing and acceptance testing.

During system testing, QA examines such things as the accounting flow, documentation, and user interface. During testing, all problems, from catastrophic errors to typographical errors, are documented in the QATR Tracking Database System.

QA tests software in *rounds* on a fixed set of media. A round begins when Software Development submits a product for a QA test cycle. QA ends a round when the QA test cycle is completed successfully, or when it cannot continue because of software problems. If a round ends because of software problems, Software Development fixes the problems based on information in the QATRs and submits the updated software to QA for another test round. QA reviews the QATRs and performs tests to verify that all the problems reported by QATRs have been fixed. If QA discovers new problems, Software Development fixes the problems and another round begins. This process continues until the software is free of known bugs.

Acceptance testing begins when the software is sent to selected beta sites for testing. At this point QA and Software Development have tested all the functional aspects of the software and fixed the major issues. The beta sites are asked to convert live databases and begin running parallel data entry. The controlled release begins when feedback from the live testing warrants further use. The QA process ends and the product is considered ready for general release when the controlled release group confirms that the software is ready for expanded use. The software is then handed off for production.

Testing Phase

Quality Assurance's participation in the product design stage has become increasingly important. Software design is often complicated and requires many different opinions to define a product that will efficiently meet the needs of many different kinds and sizes of business. That is why product suggestion forms are important.

QA uses the testing design phase to learn about the product and determine the best approach to testing the product. QA tests not only the functionality of the product, but also that the product adheres to standards which provide consistency and continuity, accommodates data flow in the application and interfacing applications, and is user-friendly.

Alpha Stage

The purpose of the alpha stage is to gain enough feedback and criticism about the design, the user interface, and the maintenance reporting aspects of the product to continue the development effort without major design changes. It begins when about one month of programming is left. Software Development drives this stage. Resellers participate in the evaluation of product specifications and the software product.

QA Determines What to Test

Before you can determine what to test, you need to understand what has changed. The project might range from simple bug fixes to a never-before-tested product, in which case determining what to test is complex.

Once you have determined the scope of the changes, determine how and where each change affects the software. To isolate the changes, first find out which programs have been affected, and then find out what those programs do. Consider whether the changes affect one another. Determine how each change affects other functions in the application or in an interfaced application.

After you have isolated the areas that require testing, determine which tests to perform. You might be able to use or modify existing tests and checklists to test functions or features of the software, or you might need to write a new test or checklist.

QA Writes the Test Plan

The QA test plan describes in concise detail the test procedures and conditions for testing the features and performance of the software product. It is a guide to help the QA analyst perform quality assurance tests after the programmer has completed adequate and substantial program testing, and before a product is released to the market. It describes the functions to be tested, the objective of each test, the requirements for each test, the conditions under which each test is to be conducted, and the procedures for each test. In addition, test plans are also used as an audit trail of the QA process.

QA test plans are based on the application's technical specification, which describes the intended function of the application. Software Development writes the specification, and QA and other departments (such as marketing and sales) review it.

Design Phase

Before testing starts, QA determines what to test and creates the necessary tests and checklists. QA then tests the software. The product is ready for beta site testing when QA determines that Software Development has completed the functional programming and that data flows through the system without critical software bugs. At this point, major software design issues have been resolved, a preliminary draft of the user's guide has been completed, and QA has tested the data file conversion programs with live data. When beta site testing starts, QA begins work on noncritical functions such as help, sample data, GENERAL Report Writer data, ODBC data dictionaries, Global Inquiry, and EIS functions.

Beta Stage

The purpose of beta testing is to completely test all functional and technical aspects of the software and to locate software bugs and verify the corrections to those bugs. A good mix of beta sites is necessary to ensure a variety of testing scenarios. Each participant in the beta testing program attends a teletraining session, and external sites complete an evaluation guide about each application by a deadline. Sites receive updated software and user's guides as needed.

Tests are easy to execute and manage if performed in logical segments. QA testing includes some or all of these tests:

- ☐ Review Paperwork
 - ☐ review older version bug reports
 - ☐ review application-specific checklists
 - ☐ review previous test plan documents
- ☐ Create New Test Plan
- ☐ Perform Fresh Install (first round)

- ___ test standalone functionality
- ___ interface other applications
- ___ enter each level of program using each menu without receiving errors
- ___ exit from each level without receiving errors
- ___ verify all functions work at a basic level
- ___ print reports to the screen, a file, print preview, and a printer
- ___ ensure HLE areas function according to standards
- ___ ensure screen commands work according to standards
- ___ complete posts in summary and detail
- ___ ensure master file data validity
- ___ test file conversion upgrades from previous to current version
- ___ Perform Consecutive-Round Test
 - ___ perform Windows/DOS file compare from previous round
 - ___ analyze a list of the program changes
 - ___ install media
 - ___ test for regression (spot-check to ensure fixes don't create other issues)
 - ___ test user interface (such as commands, windows, and consistency)
 - ___ test keys (command and in-field edit)
 - ___ test all functions
 - ___ ensure screen and report naming conventions were followed

- ___ test windows (help, inquiry and maintenance) and command bars
- ___ test messages (both errors and warnings)
- ___ test reports for accuracy, design, and naming conventions
- ___ test GENERAL Report Writer and ODBC for new or changed files
- ___ analyze sample data for accuracy, appropriateness, and completeness
- ___ test forms (ensure program prints correctly on forms)
- ___ Perform Systems Test
 - ___ test menu selections, functions, and aspects of the application
 - ___ test options and interfaces
 - ___ multiuser tests (file locks and record locks function appropriately)
- ___ Review the User Guide
- ___ Perform Minisystems Test
 - ___ include one or more accounting periods
 - ___ focus on different functions or features of the application
 - ___ use steps for consecutive round testing
- ___ Perform Acceptance Test (Beta through Controlled Release) Program

QA has standard checklists; test-case instructions; test standards for testing such things as reports, screen functionality, menus, GENERAL Report Writer data files, sample data, data dictionaries, installation, upgrades and data conversion; and paperwork review. Here are the test plan setup steps:

1. Copy your master test plan into an area where you can modify it.
2. Begin test plan development as if you were a user setting up the application.

3. Read the application specifications to select from the master test plan the test options and steps that are applicable.
4. Create the test plan based on option settings in this order: File Maintenance, Daily Work, Reports, Posts, and then Interfacing application(s).

Testing Standards

- QA performs tests according to written test plans, using the product's minimum operating environment.
- QA uses QATRs to report all problems, no matter how small. QA provides Software Development with as much information as possible about the problem, what type of testing was being done at the time the problem occurred, and where in the software the problem is located.
- QA tests the software and reads the user documentation, making sure that they agree with the technical specification.
- QA tests software in rounds on a fixed set of media. A round begins when Software Development gives the media to QA. A round ends when QA completes the test cycle or when QA cannot continue because of software problems. The process ends when all issues are resolved.
- QA backs up data files frequently, especially before posting processes, after setting up for specific tests, and before performing period-end and year-end functions. This ensures that the proper data is available to recreate a specific problem or to use for a later test.

Multiple Rounds

The second and subsequent rounds consist of regression testing and unit testing of fixed QATRs. The software goes through as many rounds as necessary to address all the issues and complete all the testing.

Copy blank data files in your application directory to refresh your data files before you install each round of media. This can ensure data integrity in the testing environment during the first few rounds but is not always necessary. Data from a previous round could cause problems in the current round. Check with the application programmer to find out whether data integrity remains good or whether data files have changed before you install new media. It may be necessary to create new data files.

Continue to perform testing procedures as you work through the test plan. Watch for errors or problems in screen appearance, functional inconsistencies, and awkward or unfriendly software usage. Make sure that standards are followed in screens, windows and reports.

Following the test plan, be sure to print reports to the printer, a file and the screen. Check each printout carefully for inconsistencies in standards, typographical errors, poor report format or meaningless column headers or detail. Check the detail printout by making sure that figures add up correctly, that column and row totals are correct, and that you can trace detail data that appears to what is contained in a file or to what you entered for purposes of testing.

When you encounter problems during testing, try to duplicate the problem, allowing yourself a good understanding of how the problem occurred. Regression-checking software helps to reenact key presses and functionality that lead to the error. Log all problems into the QATR Tracking System, regardless of how large or small.

Continue working through the test plan, performing test plan scenarios and reporting issues that need to be addressed until you complete the test plan. This signals an end to the testing rounds phase and moves on to the system test phase.

Test Evaluation

While you enter data and print reports for a test or checklist, watch the screen for errors or problems, appearance, functional consistencies and user-friendliness. Make sure that the software is working according to its specification and the standards in this document.

As you execute tests, you will print reports and screens. Check the printouts carefully for errors. To retain the integrity of the testing, check the printouts in the order of the test steps, usually from an input worksheet to a print screen (of data entry) to a report and from one or more reports to another. Always check from the correct documents; for example, always check a report or print screen with a report or print screen that you have already traced for accuracy.

Every time you execute a step in the test plan, mark it as completed. You can use a special mark in the test plan and on the accommodating printouts if necessary. This helps you keep the test results organized.

As you encounter problems, try to duplicate the problem before completing a QATR form. When filling out the QATR form, provide as much information as necessary to re-create it. Attach copies of test printouts to QATRs if you are using a handwritten form, or mark the QATR number on the test output when applicable to illustrate or document problems for the programmer. No error or problem is too small to report.

Design Issues

Design issues may arise during testing, either from a lack of understanding the functional intent or from examination of a feature or function that you feel should work differently. Ideally, QA is involved during the design phase of a project to eliminate or reduce the number of design issues that arise during the testing phase. QA's purpose is to address critical design issues, not to change design. If a design issue comes up during testing, write up a QATR form. Generally, design issues are not fixed unless they are critical to the functionality of an application; however, design QATRs should be considered later for future enhancements to an application.

Reporting Software Problems

A QATR is a standard part of a product's development cycle before and after its release. It is a medium that a tester uses to document issues (errors, suggestions, and so on) for the benefit and use of the authors who need to know about them. The online QATR system is a database in which the tester enters the information and the application programmer retrieves it.

The QATR database allows for easy storage and retrieval and reduces the chance for duplication and loss of source documents. It also gives the users a wide choice of report options. The QA tester can obtain a report of all fixed programs, which makes it easy to check the fixes on the next round of media. You can select reports based on particular criteria, reducing the amount of paper used and preventing the occurrence of duplicate problem reporting. This system also allows the programmer to retrieve QATRs that are open and in need of being addressed.

Effective communication about software problems, changes, and resolutions is important. It helps to complete the quality assurance process in a timely and thorough manner, minimize risk, and maximize confidence in the final product.

As you find problems in the software during testing, document them on the QATR form or the QATR Tracking database. The form should clearly communicate the problem to the programmer who must investigate and to the QA person who will test the QATR during the next round of media.

When you write a QATR form, attach adequate printouts supporting the problem to the form. If you use a QATR database, mark the printouts and supporting documents with the QATR reference number. Printouts help the programmer and QA analyst re-create the problem.

Depending on the nature of the QATR, you may have to rerun a test that was written for the original test round. For example, suppose that the original test was to verify that the Post Transactions function in Accounts Payable worked. The problem was that when you tried to enter transactions, you encountered a Basic error. Because it was impossible to execute the original test fully upon receipt of the specification from development, you would execute the test while testing the QATR.

Testing Documentation

One of the steps in the quality assurance process is to check the documentation. You can often do this between test rounds. Check the following items:

- consistency
- content and clarity
- sample data is correct

- sample reports use correct format and data, and that they foot and cross-foot
- menu samples are correct—correct order and copyright information
- screen samples use correct field names and sample data
- worksheets use correct field names
- field descriptions use correct field names

Run the software as you read the documentation to make sure that it works as the documentation states. Don't rely on your memory of how the software works.

Keep in mind that the documentation is for a user. Will the user be able to understand and follow the information? Does it make sense? Does it contain pertinent information? Does it answer the questions that come to mind?

When you test application upgrades and data file conversions, follow the instructions in the user's guide. It is important to know that the upgrade or conversion works as expected and that the user has enough information to perform it and maintain the integrity of his company's data.

Always look at the instructions from the user's perspective. In addition to checking that the user is able to perform the upgrade or conversion, check that the user has enough information to start up and run the software. For instance, a data file conversion may convert existing data, but the user may also have to add fields in the file before he can run the software for daily use. The user must have enough information about what is changed, added, replaced or removed.

Different users have different installations. The instructions should be specific, but general enough to fit most installations. Since other applications are often upgraded and converted at the same time, the order of installing or converting applications should be documented where necessary.

The documentation must contain backup instructions to protect the user's data and system. They should be simple and clear to make the upgrade or conversion process as easy as possible.

During the testing phase, you need to hold meetings with the writer and development personnel to discuss issues and areas of concern. Raise the issues as you find them so that all involved parties have enough time to address them. Give the writer copies of the QATRs or of the QATR database report so that he can review the QATRs for potential documentation issues.

Acceptance Testing

Beta sites selected for acceptance testing must comply with or exceed your beta site qualifications. Organizing, preparing and implementing a beta site program requires dedicated effort on the part of the QA project leader and the beta sites. Send the application specification, final-round media and user's guide to the beta sites. Provide some lead time for the beta sites to review the application specifications and the user's guide before you contact them to confirm installation.

Contact each beta site representative to confirm installation and obtain a begin date for application performance testing. It may be helpful to instruct your beta sites on testing procedures in writing or during an on-site training visit. It may be equally beneficial to let the beta site use the application independent of training. There are several ways of conducting acceptance testing at beta sites, but it is important to keep in touch regularly with the beta site representative, writing down comments, problems or other information. Continue to conduct beta site review for 60 days. Acceptance testing involves a product rollout plan that takes the product through stages. Every major product release is involved in this process. Each stage moves the product closer to general release.

Rollout Stages:

Alpha Testing

Beta Testing

Live Testing

Controlled Release

General Release

Alpha testing begins when there is about one month of programming left. Its purpose is to gain enough feedback and constructive criticism about the product design, the user interface, and the maintenance and reporting aspects of the product to continue the development effort without major design changes. Then address changes, complete coding and move to beta testing. Software Development manages this process.

Participants in the process are internal development teams and six members of our VAR channel. The VARs receive installable media and product specifications to review.

Beta testing begins when all application functions are completely coded. Product specifications are frozen and design is set. Work on such noncritical functions as help, sample data, GENERAL Report Writer data, data dictionaries, Global Inquiry, and EIS functions begins during this stage. The purpose of beta testing is to completely test all functions, options and circumstances necessary to locate bugs, and to verify that they are addressed before moving to live testing.

Participants are the internal QA team and external beta sites (equal numbers of VARs and end users). A good mix of users is necessary to ensure a variety of testing scenarios. Beta sites receive the software on a CD-ROM.

Each participant in the program must attend a teletraining session. VARs involved in the program must have access to the Open Systems web site to receive updates and notices for themselves and their user beta sites throughout the process. External sites complete an evaluators' guide by a deadline. The sites receive updated software and user's guides as needed.

Live Testing begins when all known software issues have been addressed. Its purpose is to gain a perspective of the condition of the software by using it in a live environment with daily audits and checks. Sites convert their real data and complete a checklist of a full cycle process, from file maintenance to daily work, through the completion of a month end.

Participants include beta VARs who run their businesses on OSAS, and beta users current on the Maintenance Program.

Controlled Release takes place when feedback from the live test confirms the expanded use of software. Its purpose is to get a larger user base but in a controlled environment. Ship orders to users who require new features for implementation. Control the distribution of needed corrections to a manageable group of users.

Participants are external: Executive and Partner VARs who are current on the Maintenance Program, several users who are current Maintenance Program subscribers, and several new installations receive the release. The participants must be informed that they are a part of a controlled release of the product.

When **General Release** takes place, backorders are filled, and software is shipped to users and dealers who are current on the Maintenance Program.

Media Verification Checklist

- _____ 1. Virus Check
- _____ 2. Install/Reinstall (RM only)
 - _____ A. Verify the four types of installs of Resource Manager:
 - _____ Original Install. Verify that RM installs fresh on:
_____ Mac _____ UNIX _____ Linux _____ Windows
 - _____ Reinstall through RM. Verify that RM reinstalls over itself using its own installation menu choice:
_____ UNIX _____ Linux
 - _____ Reinstall using the InstallShield installation process.
_____ Mac _____ Windows
 - _____ Reinstall RM from the operating system prompt. Verify RM installs over itself from the OS with the proper warnings:
_____ UNIX _____ Linux
 - _____ Install over old versions (if applicable):
_____ 6.5 _____ 7.0
 - _____ B. RM functionality check:
 - _____ Install an application on Windows, UNIX, Linux, and Mac.
 - _____ Create data files for RM and the application just installed
 - _____ C. Other applications:
 - _____ Verify that application installs (CD-ROM media)
 - _____ Verify the application reinstalls and gives the proper reinstall choices (CD-ROM media)
 - _____ Verify the application information to ensure that it is current
 - _____ Remove the application to ensure that it removes all programs, Report Writer data, and sample data, and that live data remains.
- _____ 3. Compares
 - _____ A. Compare a directory containing the latest maintenance fixes to the master media.
 - _____ B. Using Windows to Linux Text File Compare Utilities:
 - _____ Compare Windows to Linux text, program and data files.
 - _____ Compare Windows to Mac program and data files.
- _____ 4. Multiple Copies of Media Match:
 - _____ Verify that all copies (Offsite, Original and Working) match.

QA Terminology

Paperwork	Paperwork consists of the printouts and listings received with each media round: directory listings, file compares, label file printouts, and other reports produced with system utilities.
Project Plan	The project leader writes the QA project plan. It provides the framework for each project that the department tests. It aids in preparing the test plan and provides direction for the project. The project plan consists of these sections: title, name of the QA project leader and other participants, list of pieces, notes to other departments, and QA tasks, which are grouped into five phases: design, testing, release, evaluation, and implementation testing. Include the QA person assigned to the task, the target and actual completion dates, and the estimated hours required to complete them.
QATR	The Quality Assurance Test Report is a form used to communicate problems, design issues, or questions that QA would like the programmers to address and are stored electronically in the QATR Tracking System Database.
Regression Testing	Regression testing requires that tests done in a previous round be redone to ensure that what was working correctly in the previous round is still working correctly. It is needed most often when there are many changes in a round, particularly in critical functions such as posts.
Round	Software is tested in rounds on a fixed set of media. A round begins when Software Development submits a product for a QA test cycle. A round ends when QA completes the test cycle or cannot continue, signaling that another round of media which addresses the problems discovered during the round which just ended must be submitted. QA reviews the paperwork and retests the QATRs to verify that all the corrections were made and no new problems occur. The process ends when all issues are resolved. The round number is incremented for each new set of media when it is handed off to QA.

Sample Data	Sample data is distributed with each application so that users can learn the system without worrying about their own “live” data. During installation, it is put into its own directory, \OSAS\SAMPLE, with company ID “H”. Occasionally, sample data is used for test data, but it is not considered a good base because of the way it was initially built and because it may not be correct.
Specification	A specification is a document that describes an application’s functional and technical details. QA uses it to determine whether the product works as designed.
Surround Testing	Surround testing requires that you test events related to the function you are testing. For example, if the application’s post to General Ledger is changed, surround testing would include checking the post to verify that the other files updated at that time are updated correctly.
Systems Test	The systems test is usually done for implementations and updates, and need be only one month of processing. Systems tests are designed to test overall functionality to verify that the software still works as expected. It involves running nearly every program. Sometimes a piece of the systems test (such as a month’s or a period’s transactions) is substituted for the minisystems test.
Test Data	Test data is used in testing applications. Generally, it is the specific data designed for the systems test or the minisystems test. Sometimes sample data is used as test data, but usually only for special or miscellaneous tests. When sample data is used, it is commonly copied to the \OSAS\DATA directory to use as “live” data.
Test Plan	The test plan describes in complete and concise detail the tests and procedures that will be used in testing the features and performance of an application or a product. The test plan is a guide to aid in and a checklist for testing the product. The test plan serves an audit trail for the QA function. The test plan should include these sections (as needed): the title page, a table of contents, test objectives, reference documents, hardware and software configurations, an approach to testing, test descriptions and a checklist. The test descriptions should include enough detail to define specifically what is being tested or refer to separate tests such as the systems test.
Unit Testing	During phased testing, some pieces of a function or application are tested while the other functions are nonfunctional or incomplete. QA does not look at the nonfunctional or incomplete pieces.