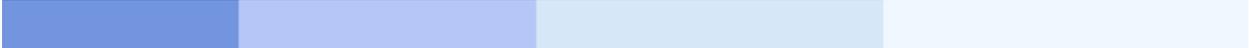




Com-plete

System Programming

Version 6.2.1



This document applies to Complete Version 6.2.1 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© March 2002, Software AG
All rights reserved

Software AG and/or all Software AG products are either trademarks or registered trademarks of Software AG. Other products and company names mentioned herein may be the trademarks of their respective owners.

Table of Contents

Overview of the System Programming Documentation	1
Overview of the System Programming Documentation	1
Terminology	2
Startup and Initialization	4
Startup and Initialization	4
Initialization - Com-plete Startup Procedure	5
Initialization - Com-plete Startup Procedure	5
MVS Procedure	5
MVS Job Control	5
MVS Required DD Statements	6
MVS Symbolic Parameters	8
Initialize Com-plete System Intercept (VSE)	9
Executing the COMSIP Program	9
VSE Job Control	9
VSE Logical Partition Assignments	10
VSE Printer Assignment	10
VSE Initialization Files	11
Com-plete Subsystems	11
Startup Options (Sysparms)	13
Startup Options (Sysparms)	13
Binary Modifications (APPLYMODS)	48
Binary Modifications (APPLYMODS)	48
Defining Terminals and Printers	59
Defining Terminals and Printers	59
Overview	59
How to Code TIBTAB	59
TIBSTART Macro	60
TIB Macro	61
CMDEVS Macro	72
TIBEND Macro	73
END Statement	73
How to Create TIBTAB	73
Dynamic Completion of TIBTAB During Com-plete Initialization	73
Printout Spooling TIBTAB Considerations	74
Defining Virtual Printers	74
Defining Real Printers	74
Batch Output Facility (SYSOUT)	74
Defining JES/POWER Nodes	75
Dynamic Routing	75
Logical Output Drivers	75
Specifying Logical Output Drivers	76
Logical Output Driver Interface	76
Return Codes	77
Line Printer Daemon (LPD) Protocol	78
Introduction	79
Concepts	79
Printing via the Local Workstation	80

Printer Definition Using Environment Variables	80
Printer Search Sequence	80
LPD Spool vs. Com-plete Spool	81
EBCDIC - ASCII Conversion	81
Logical Output Drivers	81
Printing via Printer "Boxes" Supporting the LPD Protocol	82
Changing Printer Definitions Dynamically	83
Multiple Copies of Com-plete	84
Multiple Copies of Com-plete	84
Multiple Com-pletes in One System	84
Installation Considerations	84
Sysparm Considerations	85
Multiple Com-pletes in a Parallel Sysplex	85
General	86
Shared Datasets	86
Separate Datasets	86
COMSYS Data Containers	86
Startup JCL Procedure(s)	87
Sysparm Considerations	87
Internals	88
Internals	88
Com-plete Files and Associated User Files	89
Com-plete Files and Associated User Files	89
COMSD - Com-plete Sequential/Direct Dataset	89
Dynamic SD files	90
COMSPL - Com-plete Spool Data Set	92
CAPTURn - Com-plete Capture File(s)	93
COMSYSn - Com-plete System Data Containers	94
LOAD - Distributed Load Module Library	94
USER LOAD - User Load Module Library	95
MAP Library	96
PROFILES - Editor Profiles Library	96
SOURCE - Com-plete Distributed Source Library	97
UDEBUG Profile Library	98
UDEBUG Text Card Library	98
Edit Source Libraries	99
Overview	99
Creation of Edit Source Libraries	99
Maintenance of Edit Source Libraries	101
Protection	101
Application-Specific Data Sets	101
COMDMP Dump Data Set (VSE only)	102
The System Data Infrastructure	103
The System Data Infrastructure	103
Introduction	103
Sharing Data Among Multiple Com-plete Nuclei	104
The System Data Access Method (SDAM) API	105
The SDAM Control Block	105
SDAM Views	107

The Com-plete Task Structure	108
The Com-plete Task Structure	108
Dispatching (Task Selection)	108
Thread Selection and Reservation	108
Relocation	109
The Quiesced State	109
Com-plete Resource Usage and Estimates	110
Com-plete Resource Usage and Estimates	110
Virtual and Real Storage	110
Real Storage	110
Com-plete Savepool Areas	110
Com-plete Fixed Buffer Pools	111
Storage Key of Buffer Pool Subpools	112
The Com-plete Unit of Work (CUOW)	113
Thread Groups and Sub-Groups	113
Task Groups	113
Virtual Storage Usage	114
General Buffer Pool Usage	115
The Roll Subsystem	116
Com-plete Rollout/Rollin Processing	116
Com-plete Roll Buffers	116
The Maximum Number of Rolled Out Images.	117
The Com-plete Spool Data Set	117
Data Set Structure	117
Printout Structure	117
The Com-plete Sequential/Direct Data Set	118
The UDEBUG Buffer Pool	119
The Roll Buffer	120
CPU Usage	120
Com-plete Accounting Facility	122
Com-plete Accounting Facility	122
Overview	122
User ID Accounting Block	122
SMF Records	122
The ULOG ON Record	123
The Program Termination Record	123
The Checkpoint Record	124
The ULOG OFF Record	124
The User Record	124
SMF Record Contents	125
SMF Record Common Portion	125
SMF Record Statistics Portion	126
Modifications to Com-plete Modules	129
Modifications to Com-plete Modules	129
Overview	129
Link Editing Com-plete Modules	129
Link Edit Return Codes	130
Com-plete Support Issues	130
SAGSIS Problem System	130
Problem Reporting	130
Com-plete Problem Solutions	131

Com-plete Fixes	132
The Zap Format	132
Com-plete Maintenance Updates	134
Com-plete Capture Processing	135
Com-plete Capture Processing	135
Capture Data Sets	135
Captured Data	136
Capture Records Processing	136
Com-plete Servers	137
Com-plete Servers	137
Overview	137
Server Definition	138
Server Main Routine	138
Server Initialization	138
Example:	139
Server Termination	139
Server Command Interface	140
Server Invocation	140
Server DSECTs	141
Server Request Routine	142
Software Interfaces	144
Software Interfaces	144
ACCESS	145
ACCESS	145
Adabas	146
Adabas	146
General Usage	146
ADALNK Features	147
Application Programming Interface	148
Application Programming Interface	148
Batch	149
Batch	149
Running Batch Programs	149
VTAM	151
VTAM	151
Defining and Activating the VTAM Application	151
Generic Resource names	153
LOGMODES	153
APPC Interface	154
APPC Interface	154
Concepts	154
Implementation	155
Requirements	155
CICS / Com-plete Transaction Routing	156
CICS / Com-plete Transaction Routing	156
CICS Considerations	156
Com-plete Considerations	157
Transaction Parameters	157
Logon Security	158
Programming Notes	158
TRACES	158

LIBRARIAN (MVS Only)	159
LIBRARIAN (MVS Only)	159
VSE LIBR Service	163
VSE LIBR Service	163
Using VSAM with Com-plete	164
Using VSAM with Com-plete	164
Introduction	164
VSAM Record Sharing and Integrity Options	166
Overview	166
Buffers	167
Using VSAM Files Online	168
VSAM Files	169
Alternate Indices	170
GDDM	172
GDDM	172
Installing GDDM	173
Com-plete Components for the GDDM Interface	173
Adding DFHEAI and DFHEAI0	174
VSAM, ADMF, DFHTSD	175
Altering the CICS environment	176
TIBTAB	176
Com-plete JCL Modifications	177
Com-plete Startup Parameters	177
UUTIL/ULIB Activities	178
User Calls	178
Execution	179
Performance Considerations	179
PANVALET (MVS only)	180
PANVALET (MVS only)	180
Installation Overview	180
Installation Procedure (MVS)	180
Job Entry Subsystem (JES) Interface Modules	184
Job Entry Subsystem (JES) Interface Modules	184
MVS JES Interface Modules	184
JES3	185
JES2/JES3 Server Commands (only for the old interfaces JES2SERV and JES3SERV)	185
Extended Console Server	186
Syntax:	186
Example:	187
UDS VSAM SERVICES (MVS Only)	188
UDS VSAM SERVICES (MVS Only)	188
Installation Considerations	188
Operation Considerations	189
UDVS VSAM SERVICES (VSE Only)	190
UDVS VSAM SERVICES (VSE Only)	190
Installation Considerations	190
Operation Considerations	190
UDS VSAM SERVICES (MSP/FACOM Only)	191
UDS VSAM SERVICES (MSP/FACOM Only)	191

Security Systems	192
Security Systems	192
Modes of Operation	192
External Security System Operation	192
Com-plete Security Operation	193
External Security with COMSEC	193
Interface To Natural Security	193
Defining Com-plete to ACF2	193
ACF2 v. 5.2 and lower:	194
ACF2 v. 6 and above:	194
Defining Com-plete to RACF	194
Defining Com-plete to TOP SECRET	194
The DB2 Interface	196
The DB2 Interface	196
Natural	197
Natural	197
Installing the Natural Buffer Pool Manager	197
Natural Batch	197
CA-DYNAM from Computer Associates (VSE only)	198
CA-DYNAM from Computer Associates (VSE only)	198
IBM Language Environment Considerations	199
IBM Language Environment Considerations	199
Saving Thread Storage	199
Receiving IBM Language Environment Runtime Messages and Dumps	200
VSE only:	200
File Transfer	201
File Transfer	201
UEDIT	201
IND\$FILE (MVS only).	201
Security and User Exit Facilities	202
Security and User Exit Facilities	202
Introduction	204
Introduction	204
Summary of Available Exits	204
Areas of Exit Usage	206
ULOG ON Security	207
SYSCOM,SYSNAT	208
Batch/TPF User IDs	208
ULOGX1 Exit	208
Program, SD File, File I/O Security	209
Control Programs	209
Message Switching and Printout Spooling	209
Utility and Application Security	210
Utility Security	210
Application Security	210
COM-PASS Security System	211
ACCESS User Exits	211
User Exit Considerations by Type of Exit	212
User Exit Considerations by Type of Exit	212
Batch Exits	212
Thread Exits	212

Nucleus Exits	213
Creating or Modifying a User Exit	214
Creating or Modifying a User Exit	214
ACSUUEX1 - ACCESS Write-Intercept Exit	215
ACSUUEX1 - ACCESS Write-Intercept Exit	215
How to Create ACSUUEX1	215
How to Use ACSUUEX1	215
ACSUUEX1 Conventions	216
ACSUUEX2 - ACCESS Read-Intercept Exit	217
ACSUUEX2 - ACCESS Read-Intercept Exit	217
How to Create ACSUUEX2	217
How to Use ACSUUEX2	217
ACSUUEX2 Conventions	217
SDAMSEX1 - SDAM API Security Exit	219
SDAMSEX1 - SDAM API Security Exit	219
SDAMSEX1 Conventions	219
TUDUEX1 - Select Dumps by User-Defined Criteria	220
TUDUEX1 - Select Dumps by User-Defined Criteria	220
How To Use TUDUEX1	220
TUDUEX1 Conventions	220
UCOEX1 - UCOPY User Exit	222
UCOEX1 - UCOPY User Exit	222
How to use UCOEX1	222
UCOEX1 Conventions	222
UDMPX1 - UDUMP Security Exit	224
UDMPX1 - UDUMP Security Exit	224
How to Use UDMPX1	224
UDMPX1 Conventions	224
UDSEX1 - UDS Security Exit (MVS Only)	226
UDSEX1 - UDS Security Exit (MVS Only)	226
How to Use UDSEX1	226
UDSEX1 Conventions	226
UDVSX0 - Usage Control of UDS/UDVS VSAM SERVICES	229
UDVSX0 - Usage Control of UDS/UDVS VSAM SERVICES	229
How to Use UDVSX0	229
UDVSX0 Conventions	229
UDYEX1 - Control Dynamic Allocation/Deallocation of Datasets using UDDYN (MVS only)	231
UDYEX1 - Control Dynamic Allocation/Deallocation of Datasets using UDDYN (MVS only)	231
How to use UDYEX1	231
UDYEX1 Conventions	231
UEDTB1 - Library Code Table	233
UEDTB1 - Library Code Table	233
How to Use UEDTB1	233
The CMEDTB1 Macro	233
ULHMX1 - Hello Message Exit	237
ULHMX1 - Hello Message Exit	237
How to Use ULHMX1	237
ULHMX1 Conventions	237
ULMSBTCH - Batch Output User Exit	239
ULMSBTCH - Batch Output User Exit	239
How to use ULMSBTCH	239

ULMSBTCH conventions	239
Using AFP printers	241
ULMSDISK - Dynamic Printer Allocation User Exit	242
ULMSDISK - Dynamic Printer Allocation User Exit	242
How to Use ULMSDISK	242
ULMSDISK Conventions:	242
ULINUSER - Com-plete Initialization Exit	244
ULINUSER - Com-plete Initialization Exit	244
How to Use ULINUSER	244
ULINUSER Conventions	244
ULOGX1 - ULOG Security Exit	245
ULOGX1 - ULOG Security Exit	245
How to Use ULOGX1	245
Compatability with ULGEX1	245
ULOGX1 Conventions	245
ULOPADAB - Adabas User Exit	248
ULOPADAB - Adabas User Exit	248
How to Use ULOPADAB	248
ULOPADAB Conventions	248
ULSRMPEX - Modify PF Key Codes	250
ULSRMPEX - Modify PF Key Codes	250
ULSRPSFS - User-Written Service Routine	251
ULSRPSFS - User-Written Service Routine	251
How to Use ULSRPSFS	251
ULSRPSFS Conventions	252
ULSRRJE - Remote Job Entry User Exit	256
ULSRRJE - Remote Job Entry User Exit	256
How to Use ULSRRJE	256
ULSRRJE Conventions	257
ULSRSEC - User-Written Service Routine	259
ULSRSEC - User-Written Service Routine	259
Initialization Overview	259
Initialization Processing	259
How to Use ULSRSEC	260
ULSRSEC Conventions	262
UMSEX1 - UM Security Exit	265
UMSEX1 - UM Security Exit	265
How to use UMSEX1	265
UMSEX1 Conventions	265
USTKX1 - USTACK User Exit	267
USTKX1 - USTACK User Exit	267
Using USTKX1	267
USTKX1 Conventions	267
USTRE1 - USTOR User Exit	269
USTRE1 - USTOR User Exit	269
How to Create USTRE1	269
How to Use USTRE1	270
USTRE1 Conventions	270
UTMEX1 - Timer User Exit	272
UTMEX1 - Timer User Exit	272
How to Use UTMEX1	272

UTMEX1 Conventions	272
UTMEX2 - Timer Monitor User Exit	274
UTMEX2 - Timer Monitor User Exit	274
How to Use UTMEX2	274
UTMEX2 Conventions	274
UTMEX3 - Timer Monitor RJE Exit	276
UTMEX3 - Timer Monitor RJE Exit	276
How to Use UTMEX3	276
UTMEX3 Conventions	277
UUEDEX - UED Security Exit	279
UUEDEX - UED Security Exit	279
How to Use UUEDEX	279
UUEDEX Conventions	280
UUMAX1 - UMAP Initialization Exit	283
UUMAX1 - UMAP Initialization Exit	283
How to Use UUMAX1	283
UUMAX1 Conventions	283
UUMAX2 - UMAP Command Exit	285
UUMAX2 - UMAP Command Exit	285
How to Use UUMAX2	285
UUMAX2 Conventions	285
UUMAX3 - UMAP Termination Exit	286
UUMAX3 - UMAP Termination Exit	286
How to Use UUMAX3	286
UUMAX3 Conventions	286
UUPDX1 - UPDS Security Exit (MVS Only)	288
UUPDX1 - UPDS Security Exit (MVS Only)	288
How to Use UUPDX1	288
UUPDX1 Conventions	289
UUQEX1 - UQ Security Exit	290
UUQEX1 - UQ Security Exit	290
How to Use UUQEX1	290
UUQEX1 Conventions	291
UUSEX1 - USDLIB Security Exit	294
UUSEX1 - USDLIB Security Exit	294
How to Use UUSEX1	294
UUSEX1 Conventions	294
UUSPL0 - USPOOL Command Exit	296
UUSPL0 - USPOOL Command Exit	296
How to Use UUSPL0	296
UUSPL0 Conventions	296
UUSVX1 - USERV Security Exit (VSE Only)	299
UUSVX1 - USERV Security Exit (VSE Only)	299
How to Use UUSVX1	299
UUSVX1 Conventions	299
UUTEX1 - UUTIL Security Exit	301
UUTEX1 - UUTIL Security Exit	301
How to Use UUTEX1	301
UUTEX1 Conventions	301

UXEEX1 - UEDIT Initialization Exit	303
UXEEX1 - UEDIT Initialization Exit	303
How to Use UXEEX1	303
UXEEX1 Conventions	304
UXEEX2 - UEDIT Command/Termination Exit	306
UXEEX2 - UEDIT Command/Termination Exit	306
How to Use UXEEX2	306
UXEEX2 Conventions	307
UXEEX3 - UEDIT RJE Exit	308
UXEEX3 - UEDIT RJE Exit	308
How to Use UXEEX3	308
UXEEX3 Conventions	309
UXEEX4 - UEDIT LIBRARIAN/PANVALET Exit	311
UXEEX4 - UEDIT LIBRARIAN/PANVALET Exit	311
How to Use UXEEX4	311
UXEEX4 Conventions	311
UXEEX5 - Locate Exit	313
UXEEX5 - Locate Exit	313
How to Use UXEEX5	313
UXEEX5 Conventions	313
Batch Utility Programs	314
Batch Utility Programs	314
TUBATEST - Batch Interface Test Program	315
TUBATEST - Batch Interface Test Program	315
How to use TUBATEST	315
Control Card Input	316
TUDUMP - Dump Print Utility Program	318
TUDUMP - Dump Print Utility Program	318
How to Use TUDUMP	318
Parameter Input	319
TUDUMP Conventions	320
TUFILE - File Status (ONLN/BTCH) Switching Facility (MVS only)	321
TUFILE - File Status (ONLN/BTCH) Switching Facility (MVS only)	321
How to Use TUFILE	321
Parameter Input	321
Condition Codes / Return Codes	322
TULIB - Program Catalog Maintenance Utility	323
TULIB - Program Catalog Maintenance Utility	323
How to Use TULIB	323
Control Card Input	324
TULIB Conventions	324
TUMSUTIL - Printout Spool Files Print Utility, COMSPL, Backup And Restore Utility	326
TUMSUTIL - Printout Spool Files Print Utility, COMSPL, Backup And Restore Utility	326
Additional Details of TUMSUTIL Facilities	326
Parameter Input	327
How To Use TUMSUTIL	328
Control Card Input	329
Description of the Options in Detail	331
Examples	333
TUMSUTIL conventions	333

TUSACAPT - Capture File Initialization Utility	334
TUSACAPT - Capture File Initialization Utility	334
How To Use TUSACAPT	334
TUSACAPT conventions	334
TUSDUTIL -SD File Maintenance Utility	335
TUSDUTIL -SD File Maintenance Utility	335
Initialization of the Com-plete SD Data Set	335
Parameters:	336
Backup and Restoration of SD Files	337
Parameter Values:	338
DD / DLBL or TLBL Statements:	338
Return Codes	339
Miscellaneous Tables and Control Blocks	340
Miscellaneous Tables and Control Blocks	340
ULOG Info Table	341
ULOG Info Table	341
ULSODDT1 - SYSPRINT Routing Table	343
ULSODDT1 - SYSPRINT Routing Table	343
Overview	343
How to Code ULSODDT1	343
Example:	344
ULPGMTAB - The Permanent Program Table	345
ULPGMTAB - The Permanent Program Table	345
Functional Overview	345
Building ULPGMTAB	345
Activating ULPGMTAB	346
Terminal Device Type Codes	347
Terminal Device Type Codes	347
UED Edit Control Block	349
UED Edit Control Block	349
UED Pseudo-Open Control Block	351
UED Pseudo-Open Control Block	351
UEDTB1 Entry DSECT	352
UEDTB1 Entry DSECT	352
UPDTB1 Information Control Block	353
UPDTB1 Information Control Block	353
3101 Terminal Support	355
3101 Terminal Support	355
Operation	355
Keyboard Functions	355
Setup Switches	356
Programming Considerations	358
System Programming Considerations	359
VTAM Logmodes	360
VTAM Logmodes	360
MODENT Parameters	360
Sample LOGMODE Specifications	363

Overview of the System Programming Documentation

This documentation is a guide for system programmers who are maintaining Com-plete.

This documentation discusses maintenance for all operating systems supported by Com-plete. Throughout this documentation, distinctions will be made on the basis of operating systems. References to MVS include OS/390, and VSE will refer to VSE/ESA 2.1 and above. The difference between operating systems relates to:

- Differences in terminology;
- Differences in job control;
- Differences in implementation;
- Facilities not available in all environments.

If you are planning an upgrade to a new release of their operating system, please contact the Customer Support Center for operating system release-specific considerations.

The system programmer's documentation provides the following information:

●	Startup and Initialization	A description of the Com-plete startup procedure Definitions of the initialization parameters (SYSPARMS) used to tailor Com-plete to an installation's requirements. A description of available system modifications (APPLYMODS). How to define terminals and printers. A description of the use of multiple copies of Com-plete at an installation.
●	Internals	A description of: - Com-plete Files and Associated User Files - the system data infrastructure - the Com-plete task structure - resource usage and estimates - the accounting facility - modifying Com-plete modules - capture processing - Com-plete servers - UEDIT: Functional Design
●	Software Interfaces	A discussion of the considerations when using Com-plete in various software environments.
●	Security and User Exits	A description of the security and user exit facilities provided for setting restrictions and controlling the use of the facilities, programs, and functions of Com-plete
●	Batch Utilities	Summaries of the batch utility programs available with Com-plete.
●	Miscellaneous Tables and Control Blocks	Illustrates various tables and control blocks for your reference.

Terminology

Differences in terminology are addressed by defining Com-plete terms and then consistently referring to the Com-plete terms. For a few frequently used terms, an '/' will be used to distinguish between operating-specific nomenclature.

Bearing this in mind, the following terms are used in this document:

DD/DLBL

for the MVS DD and the VSE DLBL statements

SYSIN/SYSIPT

for the MVS SYSIN or the VSE SYSIPT file

SYSPRINT/SYSLST

for the MVS SYSPRINT or VSE SYSLST file

CUU

Refers to the channel and unit of a device. CUU is referred to as a device number in MVS.

Job Control

Refers to the Job Control Language (JCL) in MVS and Job Control Statements (JCS) in VSE.

Load Library

Refers to an MVS partitioned data set containing load modules created by the linkage editor for loading and execution. Load library refers to a VSE library.

STEPLIB

Refers to the MVS load library from which programs can be loaded for execution. This documentation will refer to STEPLIB, JOBLIB, and system link library as STEPLIB, unless otherwise specifically noted. STEPLIB refers to the VSE libraries defined in the permanent or temporary LIBDEF search chain.

Startup and Initialization

This part of the Com-plete System Programming documentation explains the initialization procedure on supported platforms and describes the definitions and parameters that determine Com-plete's behavior when running.

This information is organized under the following headings:

- Initialization - Com-plete Startup Procedure
- Startup Options (Sysparms)
- Binary Modifications (APPLYMODS)
- Defining Terminals and Printers
- Multiple Copies of Com-plete

Initialization - Com-plete Startup Procedure

This chapter describes the Com-plete initialization procedure under the following headings:

- MVS Procedure
 - MVS Required DD Statements
 - MVS Symbolic Parameters
 - Initialize Com-plete System Intercept (VSE)
 - VSE Job Control
 - VSE Logical Partition Assignments
 - VSE Printer Assignment
 - VSE Initialization Files
 - Com-plete Subsystems
-

MVS Procedure

For MVS, initialize or start Com-plete by invoking the procedure COMPLETE. When Com-plete is initially installed, this procedure is added to the installation's system procedure library, SYS1.PROCLIB, or any user-defined procedure library.

The following figure illustrates a typical COMPLETE procedure that might be installed for an MVS installation.

MVS Job Control

During the installation process, you will either alter the supplied COMJCL procedure to suit your requirements, copy it to an installation procedure library, or use it as the basis for a job to be submitted. The following is the procedure in question. The illustrated procedure serves as the basis for the various descriptions and explanations that follow.

```
//COMPLETE PROC PREFIX='COM',
//      SYSPARM=SYSPARM,
//      OPARM=,
//      REG=6000K
//*
//*      Com-plete SYSTEM STARTUP PROCEDURE FOR OS.
//*
//* BEFORE STARTING THIS YOU MUST CHANGE THE STARTUP PROGRAM NAME TO
//* THAT OF THE TLINXX MODULE YOU COPIED TO THE APF AUTHORISED LIBRARY
//* IN THE INSTALLATION JOB #1.
//*
//* FOR A SYSPLEX INSTALLATION, ADD THE RLS PARAMETER TO COMSYS1-4
//*
//IEFPROC EXEC PGM=TLINOS,    <----- SEE NOTE ABOVE
//      PARM='&OPARM',
```

```
//          REGION=&REG,TIME=1440,DPRTY=(14,14)
// *
//*****
//STEPLIB DD DISP=SHR,DSN=AN.APF.AUTHORISED.LIBRARY
//*****
// *
//COMPLIB DD DISP=SHR,DSN=&PREFIX.USER.LOAD
//          DD DISP=SHR,DSN=&PREFIX.LOAD
//          DD DISP=SHR,DSN=&PREFIX.MAPS
//          DD DISP=SHR,DSN=APS.LOAD
// *          DD DISP=SHR,DSN=THE.CURRENT.ADABAS.SM.LOAD
// *
//COMSYS1 DD DISP=SHR,DSN=&PREFIX.COMSYS.BASE           ,RLS=NRI
//COMSYS3 DD DISP=SHR,DSN=&PREFIX.COMSYS.USERDEF       ,RLS=NRI
//COMSYS4 DD DISP=SHR,DSN=&PREFIX.COMSYS.CATALOG      ,RLS=NRI
// *
//SYSPARM DD DISP=SHR,DSN=&PREFIX.USER.SOURCE(&SYSPARM)
//SYSMAP  DD DISP=SHR,DSN=&PREFIX.MAPS
//COMSPL  DD DISP=SHR,DSN=&PREFIX.SPOOL
//COMSD   DD DISP=SHR,DSN=&PREFIX.SD
// *CAPTUR1 DD DISP=SHR,DSN=&PREFIX.CAPTUR1
// *CAPTUR2 DD DISP=SHR,DSN=&PREFIX.CAPTUR2
//SYSMDUMP DD DISP=OLD,DSN=&PREFIX.SYSMDUMP
//SYSPRINT DD SYSOUT=X
//SYSRDR1 DD SYSOUT=(X,INTRDR)
```

The COMPLETE procedure can be invoked from the operator's console via an MVS START command:

```
S COMPLETE,...
```

or via an MVS batch job:

```
//COMPLETE JOB .....
//IEFPROC EXEC COMPLETE,...
```

In either situation, an understanding of the DD statement functions and usage of the available startup options is required to implement the features of Com-plete.

The following sections use the above example as a basis for defining the Com-plete initialization procedure for MVS.

MVS Required DD Statements

The required and optional DD statements in the above procedure are described in more detail below:

STEPLIB

Required

This identifies the authorized load library on which the Com-plete MVS startup module TLINOS resides. No other Com-plete modules need to be available in this library

This data set is only referenced once during initialization and therefore its placement is not an issue.

COMPLIB

Required

This identifies the PDS library concatenation that effectively becomes the STEPLIB for the duration of the run. This means that all modules loaded during the execution of Com-plete are loaded from this concatenation.

With applymod 79 or 80 set, this will be a highly accessed and performance-critical data set. Without these applymods, the activity on this data set will depend on the number of MVS loads issued from the installation's applications.

COMSYSn

Required

This identifies the VSAM data sets containing various system information required by Com-plete. They will always be highly accessed data sets and should therefore be placed accordingly.

SYSMDUMP

Optional

If not specified, no support can be provided for any problem, as no diagnostic information is available. It identifies where the MVS control program should write a formatted dump in case of an abend or if Com-plete requests such as dump. You are recommended to specify this instead of the SYSABEND or SYSUDUMP DD statements, as more information is available for diagnosis if a problem should occur. You will also note that normally a SYSMDUMP will not take as long to occur. Estimates for the size of the data set specified by this DD statement must be made according to the IBM documentation.

CAPTURn

Optional

Each CAPTURn DD name eg. CAPTUR1,CAPTUR2 etc.. must point to a suitably created VSAM Capture Data Set when capture is being used in the system. When Capture is being used and these data sets are being accessed, the workload on them depends on the amount of data being captured. The data sets must also be placed in such a way that one does not interfere with another, because when one data set is full, it will be unloaded and reinitialized while another capture data set is in use.

COMSPL

Required

This must be used to point to the VSAM COMSPL data set. Refer to the COMSPL description in the chapter *Com-plete Files and Associated User Files* for more details.

COMSD

Required

This must point to the VSAM SD file dataset defined and initialized by TUSDUTIL for this Com-plete.

SYSPRINT

Optional

At termination, or using the STATS operator command, statistics are printed to this DD. If the DD is not specified, it will be allocated using dynamic allocation as a sysout data set.

SYSRDRn

Optional

This DD is only necessary if you wish to use the Remote Job Entry (RJE) facilities of Com-plete. It must be assigned to an JES internal reader as in the above example. Depending on the level of RJE activity, up to 9 statement can be specified: SYSRDR1, SYSRDR2 etc..

SYSMAP

Required for the UMAP utility.

Identifies the MVS load library into which UMAP will store maps. This library must also be specified in COMPLIB, so that maps can be loaded from it.

SYSPARM

Required

Identifies the file or library member in which the desired Com-plete system parameters are to be found.

If Com-plete is to be periodically stopped and started in order to test various startup options (for example, number of threads, size of threads, different TIBTAB, etc.), the symbolic parameter &SYSPARM can be used to identify the member containing the desired options. However, a short-term test of a specific option can be affected by use of the PARM option at startup time.

The various system parameters available are described in the section *Startup Options (Sysparms)*.

MVS Symbolic Parameters

You can modify the COMPLETE procedure, including the format and usage of the symbolic parameters. However, the symbolic parameters indicated below are generally sufficient to meet the needs of most installations:

&OPARM

Specifies a character string which is passed to the Com-plete control program via the PARM sysparm (see the section on the startup options for use of this feature).

&SYSPARM

Specifies the member name in the library identified by the DD name SYSPARM, which contains the control statements specifying the startup parameters (sysparms).

You can create multiple members to allow tailoring of Com-plete initialization to meet the specific conditions defined by the control statements.

&PREFIX

Specifies the default high level index (prefix) under which the Com-plete files are cataloged.

Initialize Com-plete System Intercept (VSE)

In a VSE environment, communication between the Com-plete nucleus and the user program is handled with the Com-plete SVC (SUPERVISOR CALL, normally 200).

The program COMSIP is used to dynamically install the SVC without an IPL. COMSIP requires a prior SET SDL for the SVC, and therefore must run in the BG partition.

You only need one system intercept module COMSIP for all Com-pletes running in your system.

Executing the COMSIP Program

Important:

If you have software from CA and/or Macro 4 installed, this step must run AFTER the initialization of the CA system and BEFORE the initialization of Macro 4.

Insert the following (or the equivalent) in the ASI BG JCL procedure immediately before the START of the POWER partition so that the Com-plete SVC will be installed automatically during each IPL.

```
// DLBL      SAGLIB, '.....LIBRARY'
// EXTENT   SYS010, vvvvvvv
// ASSGN    SYS010, DISK, VOL=vvvvvvv, SHR
// LIBDEF   PHASE, SEARCH=SAGLIB.APSvrs, TEMP
SET SDL
CRSVATBL, SVA
/*
// UPSI     00000000
// EXEC     COMSIP, SIZE=AUTO
```

See also the section **Startup Procedure in VSE Installation** in the Com-plete Installation documentation.

VSE Job Control

For VSE, Com-plete is initialized by running a job that can be left in the POWER reader using a job disposition of LEAVE (DISP=L). The following figure illustrates a typical Com-plete procedure that might be used at a VSE installation.

```
* $$ JOB JNM=JOB COM51, CLASS=2, DISP=D, LDEST=(,????)
* $$ LST CLASS=A, DISP=H, RBS=1000
// JOB JOB COM51  STARTUP FOR Com-plete V51
// OPTION PARTDUMP, NOSYSDMP
// ASSGN  SYS009, SYSLST
```

```

* $$ LST   DISP=D,CLASS=M,LST=SYS009,DEST=(,????)
// LIBDEF PHASE,SEARCH=(SAGLIB.COMvrs,SAGLIB.APSvrs,SAGLIB.ADA???,
                        SAGLIB.COMUSER,
                        IJSYSRS.SYSLIB,PRD1.BASE,PRD2.PROD),TEMP
// LIBDEF *,CATALOG=SAGLIB.COMUSER
/*
/*
// DLBL   COMCAT,'????????',,VSAM
/*
/*       Com-plete COMSYS CONTAINERS
/*
// DLBL   COMSYS1,'COM.COMSYS.BASE',,VSAM,CAT=COMCAT
// DLBL   COMSYS3,'COM.COMSYS.USERS',,VSAM,CAT=COMCAT
// DLBL   COMSYS4,'COM.COMSYS.CATALOG',,VSAM,CAT=COMCAT
/*
/*       Com-plete SD-FILE
/*
// DLBL   COMSD,'COMPLETE.VSAM.SDFILE',,VSAM,CAT=COMCAT
/*
/*       Com-plete PRINTOUT-SPOOL DATA SET
/*
// DLBL   COMSPL,'COM.VSAM.SPOOL',,VSAM,CAT=COMCAT
/*
/*       Com-plete VSAM DUMPFIL
/*
// DLBL   COMDMP,'COM.VSAM.DUMPFIL',,VSAM,CAT=COMCAT
/*
/*       Com-plete CAPTURE FILES
/*
/* DLBL   CAPTUR1,'COM.VSAM.CAPTURE1',,VSAM,CAT=COMCAT
/* DLBL   CAPTUR2,'COM.VSAM.CAPTURE2',,VSAM,CAT=COMCAT
/*
// UPSI   00000000
// EXEC   TLINSP,SIZE=AUTO
*
*       Com-plete SYSTEM INITIALIZATION PARAMETERS
*
/&
* $$ EOJ

```

The following sections discuss important considerations to be made when initializing Com-plete for VSE. The initialization parameters are defined in the section *Startup Options (Sysparms)*.

VSE Logical Partition Assignments

You must ensure that a logical partition assignment(s) (VSE LUB) is made for each disk CUU to be used by the Com-plete utilities USERV, UDD and UDZAP.

VSE Printer Assignment

The USPOOL utility has the ability to route a printout from the Com-plete online spool to a system printer (controlled by POWER). In order to separate this output from the Com-plete job log, a second printer definition must be made in the Com-plete startup deck. The following figure illustrates sample JCL, including the required assignments:

```

* $$ JOB JNM=COMPLETE,CLASS=2,DISP=L
* $$ LST CLASS=A,DISP=D
// JOB COMPLETE
// OPTION ...
...
// ASSGN SYS009,FEF
* $$ LST CLASS=A,DISP=D,LST=SYS009

```

Note that SYS009 is used by the Com-plete spooling task. All other parameters are installation-dependent.

VSE Initialization Files

The VSE initialization files are summarized in the following table.

File Name	File ID	LUB	Description
CAPTUR _n	COMPLETE.CAPTURE (optional)	VSAM	Com-plete CAPTURE
COMSPL	COMPLETE.SPOOL	VSAM	Com-plete spool file
COMSD	COMPLETE.SD		Com-plete SDLIB file
COMSYS1	COMPLETE.BASE	VSAM	Com-plete System Data Container
COMSYS3	COMPLETE.USERS	VSAM	Com-plete System Data Container
COMSYS4	COMPLETE.CATALOG	VSAM	Com-plete System Data Container
COMDMP	COMPLETE.DUMP	VSAM	Com-plete Dump File in case of abend

Com-plete Subsystems

The Com-plete nucleus routines are grouped into functional units, so-called subsystems. A number of basic subsystems is required to operate Com-plete, others are optional (meaning they are only required when certain functions within Com-plete are to be used). The following table lists all currently available subsystems and their attributes:

Subsystem	Function	Basic/Option	Act
ACCESS	Host and batch communication	Option	Yes
CAPTURE	Data logging facility	Option	Yes
COMSEC	Com-plete Security extension (add-on product)	Option	No
DEBUG	Application debugging aid	Option	Yes
MSGPO	Message switching / Printout spooling	Option	Yes
VSAM	VSAM file control system	Option	Yes
VTAM	VTAM interface	Option	Yes

Subsystems with "Yes" in the *Act* column are active by default. Optional subsystems not active by default can be enabled with the sysparm SUBSYS-ACTIVATE=. Subsystems active by default can be disabled with the sysparm SUBSYS-IGNORE=.

Disabling a subsystem affects the operation of Com-plete as follows:

1. Sysparms relevant to the particular subsystem are ignored.
2. Modules for that subsystem are NOT loaded during initialization. Therefore, functions carried out by the subsystem are not available until Com-plete is restarted with this subsystem active.
3. Main storage requirements decrease because the subsystem's modules are not loaded.

Startup Options (Sysparms)

The startup options, whether specified as PARM parameters in MVS or entered as statements read from SYSPARM/SYSIPT, are available as keyword parameters (socalled "sysparms"). These parameters are interpreted and processed by the Com-plete PARM-processor module at Com-plete initialization time. Note that the sysparms must be entered according to established keyword coding conventions.

When read as statements from SYSPARM/SYSIPT, each statement must begin in column one. A maximum of 80 characters per statement is allowed. More than one sysparm is allowed per statement, but successive sysparms must be separated by a comma, and the statement itself must be terminated by a blank. For example:

```
KEYWORD1=value1,KEYWORD2=value2... ,KEYWORD9=value9
```

Continuation statements are allowed: a statement in parentheses may be wrapped after a comma. For example:

```
KEYWORD1=(value1, comment: this statement is continued on the next line value2)
```

Multiple statements for the same keyword are permissible. Depending on the keyword, specifying the same keyword again may override a previous specification (example: PATCHAR); or add another member to a list (example: RESIDENTPAGE).

When entered as PARM parameters in MVS, standard PARM entry conventions apply. Each keyword must be entered in its entirety in any given statement in the format:

```
KEYWORD=value
```

Software AG recommends that you always use the full spelling, to prevent confusion with future new parameters.

In the descriptions that follow, the minimum abbreviations required for each sysparm are indicated by an underscore.

If a keyword option is omitted, the default value takes effect. If column one of any statement contains an asterisk, that statement is treated as a comment.

ACCESS-FORCE

Optional.

Value	YES NO
Default	NO

Specifies that the sign-on call in initialization will overwrite any existing entry in the Adabas SVC ID table.

ACCESS-ID

Optional.

Value	n
Default	None

Specifies the unique node number that identifies the Com-plete system.

ACCESS-LOCAL

Optional.

Value	YES NO
Default	YES

Specifies the scope of the ACCESS node (local or global).

ACCESS-NABS

Optional.

Value	n
Default	ACCESS-NABS=3

Specifies the number of attached buffers to be allocated for cross-memory services.

ACCESS-NCQE

Optional.

Value	n
Default	ACCESS-NCQE=5

Specifies the maximum number of concurrent commands for which queue space should be allocated.

ACCESS-SVC

Optional.

Value	n
Default	None

Specifies the number of the Adabas ROUTER SVC.

ACCESS-TIME

Optional.

Value	n
Default	ACCESS-TIME=30

Specifies the number of seconds before the returned response will be timed-out.

ADABAS-BP

Value	((no,key),(no,key).....(no,key))
-------	----------------------------------

Where:

no.	Is the number of elements to allocated in the buffer subpool for this key. This must be greater than 1 and less or equal than 8,192.
key	Is the storage protect key in which the buffer subpool will be allocated. This may be any number between 1 and 15. For MVS, Facom and Hitachi systems, only keys 8 to 15 should be specified here.

Default: A subpool is built for keys 8 to 15. 8,192 bytes will be allocated for each subpool and the number of areas that can exist in each subpool will be dependent on the size of the various ADALNK areas required.

This keyword is used to define the Adabas buffer pool. This buffer pool is used for Adabas interface work areas which are acquired outside of the thread *but* in the key of the thread. This parameter enables users to determine what key(s) buffer subpools will be built for and how many buffers will be in each subpool. Refer to the section about Adabas in the resource usage section of this documentation.

Notes:

1. If an error is encountered in an ADABAS-BP system parameter, the whole line is ignored. Therefore, if there is not a following ADABAS-BP specification in the system parameters, the defaults will be in effect.
2. A subsequent specification of the ADABAS-BP system parameter totally overwrites a previous ADABAS-BP specification. Therefore, if the second specification is incorrect, the defaults will again apply even if the first ADABAS-BP specification is correct.
3. If an Adabas call is issued in a key for which no subpool is built, the Adabas call will fail as there will be no subpool storage available to satisfy the request.
4. If APPC sessions or file transfer with INDSFILE will be used, the value should be set to 32767 to allow Com-plete to receive the maximum RV size defined in SNA.
5. A general BUFFERPOOL of at least the same size as VTAMBUFFER must also be specified.

Example:

ADABAS-BP=((20,9),(50,12),(100,8))

This will cause an Adabas buffer pool to be built with three subpools. The first subpool will be in key 9 and will have 20 elements, the second subpool will be built in key C(12) and will have 50 elements and the third subpool will be built in key 8 and will have 100 elements.

ADACALLS

Value	n (dbid,n)
Default	ADACALLS=10

Specifies the maximum number of Adabas calls that an application can make before the Com-plete/Adabas interface will force the application to be rolled out. This parameter is ignored if ADAROLL=NO is specified.

Note that *n* must be an integer between 1 and 32767.

Note also that *dbid* defines an Adabas data base ID, indicating that the specified ADACALLS parameter only applies to calls directed to the specified Adabas data base.

ADADBID

Specifies the default data base ID for Version 4.1 Adabas (and subsequent versions). This value is used if the application program does not supply a specific data base ID in the Adabas control block. Refer to the **Adabas Operations** documentation for a description of the use of the data base ID.

Note that *n* must be an integer between 1 and 255.

ADALIMIT

Value	n (dbid,n)
Default	4096

Note:

this parameter is ignored for attached programs.

Specifies the maximum number of Adabas calls that may be made by an online transaction without any intervening terminal I/O. Programs that exceed this limit are cancelled and error message ZAD0003 is displayed.

Note that *n* defines the maximum number of Adabas calls permitted before the program is cancelled. The maximum value that can be specified for *n* is 32767.

If ADALIMIT=0 is specified, this parameter is ignored (no limit).

Note also that *dbid* defines an Adabas data base ID, indicating that the specified ADALIMIT parameter only applies to calls directed to the specified Adabas data base.

ADAROLL

Value	n ALWAYS NO (dbid,n) (dbid,ALWAYS) (dbid,NO)
-------	--

Specifies the amount of time Com-plete will wait on Adabas calls before rolling out the program making the call.

Default: Com-plete calculates the optimum value for each database dynamically, based on the statistics for this database. The starting value is ALWAYS, i.e. at the first Adabas call, the program is always eligible for rollout. Then ADAROLL is calculated based on the average response time, using the following rule (here, A is the average response time):

$A < 0.05 \text{ sec ADAROLL}=0.1$

$0.05 \text{ sec} < A < 0.5 \text{ sec ADAROLL}=2*A$

$A > 0.5 \text{ sec ADAROLL}=ALWAYS$

Software AG recommends that you allow this parameter to default.

ADASVC5

Value	n (dbid,n)
Default	ADASVC5=13; the interface to Version 5 (or higher) Adabas will be disabled. Programs issuing a call to Version 5 (or higher) Adabas will be abnormally terminated with abend code U0004.

Specifies the decimal SVC number to be used when communicating with Version 5 Adabas (or higher).

Note that *n* must be an integer from 201 to 255 for MVS, and from 1 to 110 for VSE.

Note also that *dbid* defines an Adabas data base ID, indicating that the specified ADASVC5 parameter only applies to calls directed to the specified Adabas data base.

APPLYMOD

Value	n (n,n,...n) (m,NO)
Default	None

Specifies that the system-wide modifications *n* is to be included in this session of Com-plete. For a detailed description of each modification, refer to Binary Modifications (APPLYMODS).

Note that *n* must be an integer or string of integers between 1 and 128, separated by commas and enclosed in parentheses.

Note also that (m,NO) indicates the removal of the *m* modification.

AUTOLOGOFF

Value	NO n
Default	AUTOLOGOFF=NO

Specifies whether terminal users are to be logged off after a defined period of inactivity.

AUTOLOGOFF=NO indicates that users are not to be logged off if inactive.

Note that *n* is the number of minutes that a terminal user is allowed to be inactive before being logged off; *n* must be an integer between 1 and 600.

Specific terminals may be exempted from being logged off for inactivity by specifying "LOGOFF=NO" in the TIB macro.

Individual users can be exempted from being logged off for inactivity by specifying "YES" in the EXEMPT FROM AUTOLOGOFF field in the Com-plete user ID maintenance transaction.

BATCHLOGON

Value	YES NO
Default	NO

Specifies whether or not this Com-plete will service batch requests. See the section **Com-plete Batch** in **Migration** of the Com-plete Installation documentation for more information. Note other values than the above may be accepted but cause unusual error messages to appear at initialization.

BUFFERPOOL

Value	(Esize , Eno , Expno , Loc)
-------	-------------------------------

These values define the parameters for the building of the General Buffer Pool. For each correctly specified parameter, a subpool is built in the General Buffer Pool from which all non-specific buffer pool requests are satisfied. Please refer to the section **Main Storage Estimates** in **Resource Usage and Storage** in this documentation for more details on buffer pools.

```

Default BUFFERPOOL=( 64 , 512 , 256 , ANY )
BUFFERPOOL=( 64 , 64 )
BUFFERPOOL=( 128 , 64 , 16 , ANY )
BUFFERPOOL=( 128 , 16 )
BUFFERPOOL=( 256 , 64 , 32 , ANY )
BUFFERPOOL=( 256 , 32 , 16 )
BUFFERPOOL=( 512 , 128 , 128 , ANY )
BUFFERPOOL=( 512 , 16 , 8 )
BUFFERPOOL=( 1K , 32 , 16 , ANY )
BUFFERPOOL=( 1K , 32 , 10 )
BUFFERPOOL=( 2K , 16 , 8 , ANY )
BUFFERPOOL=( 2K , 16 , 8 )
BUFFERPOOL=( 4K , 8 , 4 , ANY )
BUFFERPOOL=( 4K , 8 , 4 )
BUFFERPOOL=( 8K , 8 , 4 , ANY )

```

Esize	Required This determines the size of each of the individual elements in this buffer subpool. The value will be rounded up to the next multiple of 64.
Eno	Required This determines the number of elements of the specified Esize that will initially be built in the buffer subpool to be defined.
Expno	Optional This determines the number of elements that the buffer subpool is expanded by if the primary Eno is not sufficient. The Expno value is also affected by the amount of space required for pre-emptive expansion of the subpool. As not all requests can expand a subpool when it becomes full, Com-plete requires that the general buffer pool expands pre-emptively. The space required for pre-emptive expansion is calculated internally. When the space available in the subpool reaches the size specified for pre-emptive expansion, the subpool is expanded by one-quarter of the number of subpool elements, or 10, whichever is lower. The Expno value must be equal to or higher than the figure used for pre-emptive subpool expansion. If the specified value is lower, it will be forced to this figure.
Loc	Optional Values: BELOW ANY (31 bit capable systems only) Default: BELOW This determines where the buffer subpool elements are to be allocated. BELOW indicates that the storage should be allocated below the 16 meg line and is the default. ANY indicates that the storage can be allocated anywhere within the primary address space and will cause it to be allocated above the 16 meg line under normal circumstances.

CAPTURE

Value	(n,REUSE/NOREUSE)
Default	Capture determines how many capture files are specified in the job control; NOREUSE

This specifies the number of capture data sets that are available for use. *n* must be a numeric between 1 and 9 and a CAPTURE DD statement /DLBL should exist for each capture data set specified. The REUSE option indicates that capture processing can reuse full data sets when all other data sets are full. The default, NOREUSE, indicates that the data set must be copied and/or reinitialized using the TUSACAPT utility.

COM-PASS

Value	YES NO
Default	COM-PASS=NO

Used to specify whether or not COM-PASS is in operation.

COMSECDB*

Value	n
Default	none

Specifies the data base ID where the Com-plete Security System data base file exists. *n* is an integer between 1 and 255.

COMSECFN*

Value	n
Default	none

Specifies the file number of the Com-plete Security System data base file. *n* is an integer between 1 and 255.

COMSECLG*

Value	YES/NO
Default	NO

Specifies that Com-plete Security violations will be logged via message ZSS0031.

* Only applicable if the Com-plete Security System is installed.

COMSTOR-BUFFERPOOL

Value	(Esize , Eno , Expno , Loc)
Default	no COMSTOR buffer pool allocated

Refer to the section **The COMSTOR Bufferpool in Resource Usage and Estimates** in this documentation for more information on specifications for this parameter.

The values have the same meaning as for the BUFFERPOOL parameter, except the default for Loc : BELOW (Non-XA systems) ANY (All XA capable systems)

DEBUG-BREAKPTS

Value	n
Default	100

Determines the initial number of areas allocated for the setting of UDEBUD breakpoints. As these areas are allocated using the Com-plete fixed length buffer mechanism, more areas are made available if required. However, this should be set to reflect the expected usage to avoid expansion and contraction of the DEBUG buffer pool.

DEBUG-GLOBALSYM

Value	n
Default	2000

Determines the initial number of areas allocated for global symbols. Please refer to the section on UDEBUD in the Com-plete Utilities documentation for details of what global symbols are and how the number required may be estimated. As these areas are allocated using the Com-plete fixed length buffer mechanism, more areas are made available if required. However, this should be set to reflect the expected usage to avoid expansion and contraction of the DEBUG buffer pool.

If an application program exceeds the CPU time specified for the thread in which it is executing, the program is abnormally terminated.

DEBUG-LOCALSYM

Value	n
Default	100

Determines the number of symbol table entries a user may define for a UDEBUD session. These are allocated out of the thread in which the UDEBUD session is running and this parameter will have a direct impact on the catalog size for UDEBUD.

DEBUG-SESSIONS

Value	n
Default	20

Determines the initial number of areas allocated to handle users running UDEBUD sessions. As these areas are allocated using the Com-plete fixed length buffer mechanism, more areas are made available if required. However, this should be set to reflect the expected usage to avoid expansion and contraction of the DEBUG buffer pool. A UDEBUD session occurs whenever the UDEBUD program is invoked. Therefore, if a user has UDEBUD running on three levels, there are three UDEBUD sessions active and three areas are required.

DEBUG-SVC

Value	Between 200 and 255
Default	255

Determines the SVC number which will be used by UDEBUD to set breakpoints. When a breakpoint is set, this SVC number replaces the applicable instruction to cause UDEBUD to get control at the appropriate point. It is a logical SVC in that the SVC must not be installed as control is obtained using the SVC screening mechanism.

It is recommended that an uninstalled SVC be used where possible, as uninstalled SVC should never normally be issued. If the SVC is issued and it is found that this is not a UDEBUD breakpoint, the SVC will be passed on to the operating system. An existing SVC could conceivably be used and would execute correctly when issued. However, each time the SVC is issued, it would incur the extra overhead of UDEBUD checking to see if it is a breakpoint and then passing on the request EVERY time the SVC is issued. For this reason, we recommend that an unused SVC number be chosen, or an SVC which is not normally issued from the Com-plete address space. The value which can be specified here must be in the user SVC number range of 200 to 255.

DYNALLOC-MSGLEVEL

(MVS only) Value	0, 4, 8, or 12
Default	12

This keyword can be used to select a severity level for messages related to dynamic dataset allocation. Messages with a severity level higher or equal to the value specified will be written to the system console and to the JES log file. Any value higher than 8 will cause Com-plete to request no messages at DYNALLOC invocation. This parameter can also be changed dynamically using UUTIL subfunction TO.

EOJ-VER

Value	character string
Default	None

Specifies that the indicated character string must be entered as part of the EOJ operator command when Com-plete is terminated.

Note that the value specified must be a one- to eight-character string.

GLOBAL-MAXENQS

Value	between 100 and 32767
Default	1024

This determines the maximum number of ENQs or LOCKs that can be outstanding from user programs in the Com-plete region or partition. Refer to the section **ENQ/LOCK Storage Area** in **Resource Usage and Estimates** in this documentation for more information about this area.

HARDCOPYKEY

Value	NONE/(interrupt key, OVERRIDE)
Default	NONE

Specifies which key is to be used to initiate hardcopy requests. Valid values for *interrupt key* are PA1, PA2, PA3, PF01, PF02 PF24, CLR and TREQ. This is the system default and can be overridden by individual users in their profiles.

The OVERRIDE option only has an effect when the selected key is PA2. If this is specified, then the application program can specify that PA2 is returned to the program rather than causing a hardcopy to be taken. If it is NOT specified, the application cannot request the PA2 key as is the case with all other keys. This option is only provided to be compatible with previous releases.

HELLOMSG

Value	YES NO dest.code
Default	HELLOMSG=NO

Specifies whether or not the Com-plete initialization hello message is to be broadcast.

HELLOMSG=YES causes the hello message to be sent to all terminals except those defined as ALL=NO in the group argument of the TIB macros.

HELLOMSG=NO suppresses transmission of the hello message.

If dest.code is specified, the initialization message is sent to all terminals defined in the TIBTAB that have the specified destination code.

INIT-PGM

Value	name (name1, name2, ..., namen)
Default	None

Specifies the name(s) of programs to be loaded by Com-plete at the end of initialization. These programs must reside in the COMPINIT. They will be started in the order *n* which they are specified, will execute in the Com-plete address space in Com-plete's key and will be deleted after execution.

Note that these programs must not use the MCALL macro.

INSTALLATION

Value	character string
Default	INSTALLATION=*****

Specifies a one- to eight-position character string used as an installation identification name. Note that a comma (,) cannot be part of this identification name.

The identification name is used by the CAPTUR function for the initial and trailer labels generated by Com-plete. It is also used in all COM-PASS utility menus.

JUMPCHAR

Value	character
-------	-----------

Specifies a character (usually a special character) as system-wide default to be used to suspend the current session with Com-plete and start the next suspended session. Users must type the character in an input field and press ENTER.

JUMPKEY

Value	key
-------	-----

Specifies a PF or PA key as a system-wide default to be used to suspend the current session with Com-plete and start the next suspended session. Possible values: PF1–PF24, PA1 or PA2.

LANGUAGE

Value	n (1..255)
Default	LANGUAGE=1 (English)

Specifies the default language used to build the user's COM-PASS menu.

LIBRARIAN

Value	''text''
Default	LIBRARIAN='NOLIST,NOEXEC' (MVS only.)

Specifies the parameters that are passed by the Com-plete editor through its interface to LIBRARIAN during a SAVE command. The values will appear on the "-SEL" card.

Note that *text* is a character string and must be enclosed within single quotation marks.

MAXENQS

Value	n
Default	MAXENQS=15

Specifies the maximum number of MVS ENQs or VSE LOCK that may be outstanding for any one application program. Each outstanding ENQ/LOCK resource held will occupy 24 bytes plus the length of RNAME in the general buffer pool while the resource is held (whether it is held as SHR or EXCLUSIVE).

Note that *n* represents any integer from 1 to 256.

MAXLIBS

Value	n
Default	MAXLIBS=40 (VSE only.)

Specifies the maximum number of entries in the VSE file table for keeping track of libraries. Each entry in the file table occupies 64 bytes.

Note that n represents any integer from 1 to 200.

MAXPOCOPIES

Value	n
Default	255

This keyword can be used to limit the number of copies a user can request when creating a printout.

n represents any integer from 1 to 255.

MAXPOLINES

Value	n
Default	MAXPOLINES=65,535

Controls the maximum number of printout spool lines which can be written to an individual printout. If this number is exceeded, the application is terminated with message MPO0203.

MAXPOQUEUE

Value	n (0 .. 32767)
Default	MAXPOQUEUE=10

Specifies the maximum number of printouts that can be queued to a printer before Com-plete reschedules to the alternative printer.

MAXPRINTOUT

Values	n
Default	MAXPRINTOUT=512

Specifies the maximum number of printouts that can be held in Com-plete's printout spool data set. You must ensure that at least as many blocks are allocated for the COMSPL data set as the MAXPRINTOUT value, otherwise I/O errors may occur.

Note that the value for n may only be changed with a Com-plete restart in which MSGSTART=COLD and POSTART=COLD are specified.

MAXTASKS

Value	n Where n is the maximum number of tasks within task groups that will be allocated. The number must be greater than zero and less than or equal to 26 for VSE (i.e. VSE maximum tasks = 32 less 6 Com-plete system tasks) or 249 for MVS, Facom and Hitachi (this is simply a nominal maximum of 256 less the 7 Com-plete system tasks).
Default	MAXTASKS=251 (MVS, Facom, Hitachi) MAXTASKS=26 (VSE)

This keyword is used to define the maximum number of tasks which will be used within a given Com-plete run.

This parameter should be allowed to default unless there is a valid reason for restricting the number of tasks to be attached. The only mechanisms for causing tasks to be attached are through the startup parameters or through the TASKS operator command.

MESSAGE-ID

Value	x
Default	Patch-character

Com-plete messages have a prefix with the format *pppgggnnnn-i*, where

ppp	Product ID (COM, TPF)
ggg	Message Group ID
nnnn	Message Number
i	System ID

By default, the patch character is used as the system ID (see the PATCHAR sysparm). Specify MESSAGE-ID=INSTALLATION to append the installation ID instead of the patch character.

MSGEXPIRE

Value	n
Default	None

The number of days a mailbox message is to be preserved.

MSGSTART

Value	COLD WARM HOT NO
Default	MSGSTART=NO

Specifies the restart option to be applied at Com-plete initialization time to the COMSPL dataset for messages not entirely written to their receiving terminals when Com-plete terminates processing (normally or abnormally). Messages with class codes of 3 will always be restarted.

MSGSTART=COLD indicates that no messages are to be requeued. In this situation, Com-plete writes zeros to each message record on the COMSPL dataset. All queued messages will be permanently lost. Note that messages are not deleted from the message file except during a COLD start or when overlaid.

MSGSTART=WARM indicates that messages not entirely written to their receiving terminals will be requeued. In this situation, all requeued messages will be restarted from their beginning except class 4, 12, 14, or 16 messages.

If a message is sent to a multiple number of terminals and not received at one or more of those terminals, it is restarted for those terminals from the beginning of the message. If a terminal user was in the process of recalling a message, it will not be restarted.

Note that when Com-plete is started with the WARM option in effect, the same TIBTAB must be used.

MSGSTART=HOT indicates that messages not entirely written to their receiving terminals are to be requeued. Printing for requeued messages is to begin from the last checkpoint taken by Com-plete.

MSGSTART=NO indicates that no messages are to be requeued. Here, none of the message records in the message queue file are destroyed. In this case, a subsequent WARM or HOT start might requeue a message originally queued before the time MSGSTART=NO was specified. Message requeuing causes the next message number to be one.

Messages with class codes 4, 12, or 14 are not requeued, and messages with class code 3 will always be restarted from the beginning.



Warning:
if a new TIBTAB is being used, you must specify
MSGSTART=COLD or the results may be unpredictable.

Note:

If the MSGSTART parameter is set to NO or COLD, the Mailbox facility will not be effective.

NATSECDB

Value	n
Default	NONE

Specifies the data base number of the Natural Security file (validates user ID and password during logon to Com-plete) *n* must be an integer between 1 and 255. See the description of the interface to Natural Security in the chapter *Software Interfaces*.

NATSECFN

Value	n
Default	NONE

Specifies the Natural Security file number (used to validate user ID and password during logon to Com-plete. *n* must be an integer between 1 and 255. See the description of the interface to Natural Security in the chapter *Software Interfaces*.

PASSWORD

Value	YES NO
Default	PASSWORD=YES

Specifies whether or not a password is required of the terminal user when the ULOG utility program is invoked with the ON option.

PASSWORD-EXPIRE

Value	n (0..999)
Default	PASSWORD-EXPIRE=0

n days after the last successful password alteration, Com-plete will force the user to change the password again.

A value of 0 advises Com-plete not to force the user to change the password at all.

PASSWORD-RETRIES

Value	n (0..255)
Default	PASSWORD-RETRIES=3

Specifies the number of attempts a user has to type in the correct password before his User ID is locked.

PATCHAR

Value	<char>
Default	"*"

Specifies a character that must, if different from asterisk (*), uniquely identify the running Com-plete within the system. If another Com-plete with the same character is active, Com-plete will be terminated during initialization. Any valid printable character can be specified as the patch character.

A patch character of "*" allows multiple Com-pletes with this patch character to be active at the same time.

This character is of importance in two areas. Firstly, every message sent to the console will have the patch character of the issuing Com-plete following the message identifier in brackets eg. ABS0006 (2) etc. Prior to the sysparms being processed, the default patch character will be shown in all messages.

Secondly, data can be added to the profile system as being specific to a certain system. When the data is read, data relating to the patch character of the running system will be searched for before taking the global information. In this way, you can customize your sessions differently in different Com-pletes using the same Com-plete system data set.

PGMLOOKASIDE

Value	name
Default	None

Specifies the name of a program to be preloaded into storage at Com-plete initialization from a library in the COMPLIB chain.

The program can be any program except those programs linked with overlay. Programs specified are accessible by terminal invocation and via the Com-plete functions ATTACH, COLINK, COLOAD, COXCTL, FETCH, LINK, and LOAD. All attributes such as RG, PV etc. are taken from Com-plete's Program Catalog. To load more than one program, multiple statements can be provided.

Note that programs residing in PDSE load libraries are not eligible for Com-plete's Fast Load Facility and are therefore ignored when specified by a PGMLOOKASIDE statement.

POEXPIRE

Value	n
Default	None

The number of days a printout is to be preserved.

POLDRV

Value	"Program Name"
Default	none

Specifies a default logical output driver name to be used for all printouts that do not specify a logical output driver themselves. For more information on logical output drivers, refer to the chapter *TIBTAB - Terminal Information Block Table*.

POSTART

Value	COLD WARM HOT NO
Default	POSTART=NO

Specifies the restart option to be applied at Com-plete initialization time to the spool dataset for printouts not entirely written to their receiving terminals at the time Com-plete terminates processing (normally or abnormally).

POSTART=COLD indicates that no printouts are to be requeued. In this situation, Com-plete writes zeros to each printout record on the spool dataset. All queued printouts will be permanently lost. Note that printouts are not deleted from the spool dataset except during a COLD start or when overlaid.

POSTART=WARM indicates that printouts not entirely written to their receiving terminals will be requeued in their entirety. In this situation, all requeued printouts will be restarted from the beginning.

Note that when Com-plete is started with the WARM option in effect, the same TIBTAB must be used.

POSTART=HOT indicates that printouts not entirely written to their receiving terminals are to be requeued. Printing for requeued printouts is to begin from the last checkpoint taken by Com-plete.

POSTART=NO indicates that no printouts are to be requeued; none of the printout records in the message queue file are destroyed. In this case, a subsequent WARM or HOT start might requeue a printout queued before that POSTART=NO was specified.

Printouts with class codes 4, 12, or 14 are not requeued, and printouts with class code 3 are always restarted from the beginning.



Warning:

If a new TIBTAB is being used, you must specify COLD, or the results may be unpredictable.

PROGRAMISD

Value	n
Default	PROGRAMISD=40

Specifies the number of In-Storage Directory (ISD) slots to be reserved for Com-plete online programs.

Each ISD entry contains the disk address of an online program that has been or is executing. For a given ISD, the entries are dynamically altered to reflect the most current program usage based upon frequency of use.

Each program ISD entry occupies 128 bytes of page-fixed storage.

Note that *n* must be an integer from 1 to 16 digits in length. The minimum value is 10.

RECALLCHAR

Value	x
Default	RECALLCHAR=NO

Specifies whether or not a recall character is available and what that character is. *x* can be any character which can be entered at a terminal. If this character is printed as the first character on ANY screen, it is taken as the recall character and the data is not passed to the application. This specification is simply the system default. Users can set their own recall characters.

The specified character, if any, must not be the same as that specified for the suspend character via the SUSPENDCHAR sysparm.

RESIDENTPAGE

Value	program-name
Default	None

Specifies the name of a program or map to be loaded and made resident at Com-plete initialization time.

Note that the program must be fully reentrant. If it is not marked reentrant, a warning message is issued on the operator's console at Com-plete initialization time.

The program or map must reside in the COMPLIB of the Com-plete initialization procedure.

The program is subsequently accessible via the COLINK, COLOAD, COXCTL, LINK, LOAD, and XCTL functions.

Maps are subsequently accessible by online applications by the use of the WRTM or READM functions.

Multiple statements can be used to load more than one program.

To effectively use UDEBUG, the majority of the UDEBUG nucleus must be loaded into the Com-plete nucleus via this parameter. A sample member DBUGSAMP is provided on the distributed source dataset containing the statements for the necessary modules.

RJE

Value	ALLOW DISALLOW
Default	ALLOW

Specifies whether or not Com-plete remote job entry requests are to be honored.

The value for this sysparm can be overridden dynamically at any time by use of the Com-plete operator commands ALLOW and DISALLOW.

ROLL-BUFFERPOOL

Value	(Esize,Eno,Expno,Loc)
Default	No fixed Roll buffer pool allocated

Please refer to the section **The Fixed Length Roll Bufferpool in Resource Usage and Estimates** in this documentation for more information on specifications for this parameter.

The values have the same meaning as for the BUFFERPOOL parameter, except the values for Loc:

Values for Loc	BELOW (Non XA capable systems)
ANY	(XA capable systems which are not ESA capable)
DS	(ESA capable systems)

The following ROLL-BUFFERPOOL SYSPARMs are generated by default:

```
ROLL-BUFFERPOOL=(064K,number_of_TIBs / 2,number_of_TIBs / 2,DS)
ROLL-BUFFERPOOL=(128K,number_of_TIBs / 4,number_of_TIBs / 4,DS)
ROLL-BUFFERPOOL=(256K,number_of_TIBs / 8,number_of_TIBs / 8,DS)
ROLL-BUFFERPOOL=(512K,number_of_TIBs / 16,number_of_TIBs / 16,DS)
ROLL-BUFFERPOOL=(biggest_THREAD_size + 8K,2,2,DS)
```

Example:

If the number of TIBs in TIBTAB is 256 and the biggest THREAD size is 1536K (512K below and THSIZEABOVE=1024), the following ROLL-BUFFERPOOL statements are generated:

```
ROLL-BUFFERPOOL=(064K,128,128,DS)
ROLL-BUFFERPOOL=(128K,064,064,DS)
ROLL-BUFFERPOOL=(256K,032,032,DS)
ROLL-BUFFERPOOL=(512K,016,016,DS)
ROLL-BUFFERPOOL=(1544K,2,2,DS)
```

VSE/ESA Considerations

If you allow the roll buffer pool to be built in a data space (the recommended way), you might need to adjust your SYSDEF DSPACE definitions as follows:

1. Increase the value of DSIZE by the size of your former ROLLBUFFER value plus the collective size of your former ROLL datasets.
2. Since each base subpool and each subpool expansion of the roll buffer pool is allocated in a separate data space, you should consider increasing the value of PARTMAX (default is 16).

Example:

Include the following SYSDEF statement in the ASIPROC for BG:

```
SYSDEF DSPACE,DSIZE=32M,PARTMAX=64
```

SAVEPOOL

Value	n>=100
Default	100

Specifies the number of "savepool" entries to be allocated for Com-plete. This is a critical parameter as these areas are used as base level save areas and can therefore not be expanded. If they are filled, Com-plete will terminate abnormally. Please refer to the section *Main Storage Estimates* in the chapter *Resource Usage and Storage* for more information.

SAVEPOOL-ANY

Value	n>=100
Default	100

This specifies the number of "savepool" entries which Com-plete should allocate above the 16 meg line. The value specified for this parameter must be carefully reviewed based on the usage of the system. When these areas run out, the system can continue to run using savepool entries allocated below the line, however, this is a waste of a valuable resource. Refer to **Resource Usage and Estimates** in this documentation for more information.

SDLEVEL

Value	NO YES userid
-------	---------------

This parameter is supported only for sites that were using it in a previous version of Com-plete. You are strongly recommended to use or omit this parameter as you did before.

SD-PREFIX

Value	dsname
Default	none

This parameter, when specified, indicates that each SD file is to be allocated as a separate VSAM dataset with a dataset name starting with this prefix. Please, refer to **Com-plete Files and Associated User Files** in this documentation for the rules describing how the resulting dataset name is built.

dsname must adhere to the operating system rules for valid dataset names. Its length must not exceed 28 characters. The last character must not be a dot ('.').

SD-RLS

Value	YES NO
Default	NO

This parameter takes effect only for dynamic SD files, i.e., when SD-PREFIX is also specified, and if DFSMS version 1.4 or higher is active on the system.

SD-RLS=YES causes Com-plete to access dynamic SD files using VSAM RLS, i.e., the dataset is allocated with parameter RLS=NRI and the ACB is opened with MACRF=RLS.

The SMS dataclass specified with SD-SMSCCLASS must allow for use of RLS access by means of the parameter LOG(NONE). Note that the LOG parameter of a dataclass does not exist in DFSMS versions prior to 1.4., therefore SD-RLS is ignored if DFSMS version 1.4 or higher is not active on the system.

SD-SMSCCLASS

Value	(dataclass,storageclass,managementclass)
Default	none

This parameter takes effect only when SD-PREFIX is also specified. It defines the SMS classes Com-plete uses for creating dynamic SD files.

Each subparameter corresponds to a DYNALLOC parameter. Com-plete does not define any default values for these parameters, i.e., when you omit a subparameter, Com-plete omits the corresponding DYNALLOC parameter causing system defaults, if any exist, to take effect.

SDSIZE

Value	n
Default	SDSIZE=1500

Specifies the size of the initial SD file that is allocated by the Com-plete utilities UED and UEDIT.

Note that *n* represents the number of records of a member being edited. Here, *n* must be an integer between 10 and 9999.

SECDEFAULT

Value	ALLOW DISALLOW
Default	SECDEFAULT=DISALLOW (MVS only.)

Specifies whether or not Com-plete will honor all password-protected file requests. This parameter is checked in module ULSRSEC, the password protection routine.

SECSYS

Value	NO RACF ACF2 TOPSECRET COMSEC,R A T
Default	SECSYS=NO

Specifies an alternate security system to be used. This subsystem is used to validate user IDs and passwords during logon and is interrogated in order to determine data set access authority during utility processing.

SECSYS-APPL

Value	name
Default	SECSYS-APPL=COMPLETE

Specifies the Application Name to be used for uniquely identifying this Com-plete nucleus to the External Security System (see SECSYS). "SECSYS-APPL=&VTAMAPPL" tells Com-plete to copy the name from the SYSPARM VTAMAPPL.

SERVER

Value	(serv-id , init-mod , p1 , p2 pn)
Default	none
serv-id	the ID for this server (1-8 chars)
init-mod	the name of the initialization/termination routine.
p1...pn	parameters to be passed to the init-routines

This will cause Com-plete to build a Server Directory Entry (SDE) for the specific Server and pass control to the initialization routine specified to cause the Server to be initialized. Refer to the chapter *Com-plete Servers* for more information.

SMFCHECKPOINT

Value	n nT
Default	SMFCHECKPOINT=15

Specifies the time in minutes or transactions, allowed to elapse before an SMF checkpoint record is written for a terminal user.

n specifies an integer from 1 to 32767 as the number of minutes to elapse before the checkpoint record is to be written.

Note that nT specifies the number of transactions (i.e., program dispatches from the ready-to-run queue commonly called a called a ROLLIN) to be executed before the checkpoint record is to be written, where n is an integer from 1 to 256, and T is a constant.

SMFLOG

Value	(r1,r2,r3,r4)
Default	SMFLOG=(LOG,NO,NO,LOG)

Specifies the logging action to be taken for the various SMF records written by Com-plete. The four positional parameters represent the following SMF records:

- r1 ULOG ON record
- r2 program termination record
- r3 checkpoint record
- r4 ULOG OFF record

Each positional parameter must be replaced with one of the following values:

- NO the corresponding record is not to be logged.
- LOG the corresponding record is to be logged to the operating system logging device (WTL/SYSLST logging device)
- OPERATOR the corresponding record is to be logged to the operator's console (SYSLST)

SMFRECORDS

Value	(r1,r2,r3,r4,r5)
Default	SMFRECORDS=(0,0,0,0,0)

Specifies the user SMF record numbers, if any, to be assigned to the corresponding Com-plete accounting records. The five positional parameters represent the following SMF records:

- r1 ULOG ON record
- r2 program termination record
- r3 checkpoint record
- r4 ULOG OFF record
- r5 User-defined SMF record

Each positional parameter must be replaced with either the number 0 or an integer from 128 to 255.

Note that 0 (zero) specifies that the corresponding record is not to be written.

Valid user SMF record numbers are 128 through 255. Replacing a parameter with a number from this range causes the corresponding accounting record to be written to the SMF file using the designated number.

In VSE, the SMF accounting records are written to the the Com-plete capture data set(s).

STACKMAXIMUM

Value	n
Default	STACKMAXIMUM=9

Specifies the maximum number of suspended transactions a COM-PASS user may have.

Here, n must be an integer from one to nine.

STARTUPPGM

Value	name (name1,name2,...,namen)
Default	None

Specifies the name(s) of programs to be invoked by Com-plete at the end of initialization. The programs specified can be any Com-plete application programs. They will be started in the order in which they are specified and will execute as attached tasks under Com-plete's user ID.

Note that there must be sufficient batch or free TIBs in Com-plete's TIBTAB to accommodate the number of programs specified.

Note:

For VSE, STARTUPPGM=U2SPIT is required in order to build the LIBRARY TABLE. Failure to do so will result in abnormal conditions for the utilities USERV, UED, and UEDIT.

SUBSYS-ACTIVATE

Value	subsystem name
Default	none

Activates a subsystem not active by default. Refer to the section *Com-plete Subsystems* above for more information.

SUBSYS-IGNORE

Value	subsystem name
Default	none

Deactivates a subsystem active by default. Refer to the section *Com-plete Subsystems* above for more information.

SUSPENDCHAR

Value	"x"
Default	SUSPENDCHAR=NO

Specifies whether or not a suspend character will be available and what that character is. x can be any character which can be entered at a terminal. Entering this character without data has the same effect as pressing the suspend key.

If this character is entered as the first character on ANY screen, it will be taken as the suspend character and the data is not passed to the application. This value is simply the system default. Users can set their own suspend characters.

The specified character, if any, cannot be the same as that specified for the recall character via the RECALLCHAR keyword.

SUSPENDKEY

Value	NONE interrupt key
Default	SUSPENDKEY=NONE

Specifies which key is to be used to suspend a COM-PASS level. Valid values for *interrupt key* are PA1, PA2, PA3, PF01, PF02 PF24, CLR and TREQ. This is the system default and can be overridden by individual users in their profiles.

TASK-GROUP

Value	(grp,no.,priority,maxq)
Default	TASK-GROUP=(DEFAULT,1)

This keyword is used to define the task groups, each containing one or more tasks, which will be available when Com-plete is started.

Where:

grp	Is the name of the task group being defined.
no	Is required and indicates the number of tasks to be allocated in the task group. This value must be greater than 1 and less than 249 (MVS) or 26 (VSE).
priority	Default: 248 is the operating system priority to be assigned to the operating system task which is attached for MVS, Facom and Hitachi systems only. This parameter will be accepted under VSE but will have no meaning. The value specified must be between 0 and 255 inclusive. Note that the highest priority which can be assigned is the priority at which the task dependent service processor task is running. Without HPE, this will be 250 therefore, while 255 will be accepted, the task will in fact only be given a priority of 250.
maxq	Default: 16 Specifies the maximum number of TIBs which are expected to be on this task groups work queue at the same time. Under normal circumstances, the default should be perfectly adequate. When there are problems and it is not, a secondary Last In First Out (LIFO) queue will be used so that no work is lost. The normal queue is First In First Out (FIFO) which insures that work is done in the order in which it is received which is why the LIFO queue is only used as a secondary backup.

Notes:

A maximum of 8 task groups may be defined.

Task group names will be upper cased prior to being processed, therefore, entering the parameter in lower case will be treated as, and appear in, upper case letters.

Where more than one specification appears for a task group, the last valid specification will be used.

The task group `DEFAULT` must always exist in the system. If it is not explicitly defined by the installation, the task group will be built by the system with the default values.

Note that the total number of tasks to be attached must not exceed the `MAXTASKS` specification. This is not checked until the task groups are being built and exceeding the value will therefore lead to task group allocation errors as against parameter errors.

Examples:

```
TASK-GROUP=(DEFAULT,4)
```

This will cause the `DEFAULT` task group to be allocated with four attached tasks and the default priority and maximum queue size specification.

```
TASK-GROUP=(DEFAULT,4,200)
```

```
TASK-GROUP=(TASK-GRP,4,150)
```

This will cause the `DEFAULT` task group to be allocated with four attached tasks with a priority of 200 and the default maximum queue size specification. A second group called `TASK-GRP` will also be allocated with three attached tasks, a priority of 150 and the default maximum queue size specification.

TELNETPORT

Value	n (1....65535)
Default	none

This keyword is used to activate the TELNET (th3270) server in Com-plete. *n* specifies a valid TCP/IP port number available to Com-plete. This port number can then be used in a terminal emulation to connect to Com-plete.

THREAD-GROUP

Value	(grp,(sub,size,no,cpu,real,key),...,(sub,size,no,cpu,real,key))
Default	THREAD-GROUP=(DEFAULT,(\$DEFAULT,128,2))

This keyword is used to define the thread groups, each containing one or more thread sub-groups and threads, which will be available when Com-plete is started.

Where:

grp	Is the name of the thread group being defined.
sub	Is the name of the sub-group being defined. If a sub-group name is specified more than once for the same group, the last valid specification will be used when parameter processing has been completed.
size	Is required and indicates the amount of storage in Kbytes which will be allocated for each thread below the line. This value must be 8k or greater and less than 1 megabyte. Each sub-group in a thread group must have a different size.
no.	Is required and indicates the number of threads to be allocated in the thread sub-group. This value must be greater than 1 and less than 4,096.
cpu	Default: 2.00 seconds Is the CPU time in seconds which a user program can use in the thread sub-group for one Com-plete transaction. This value may be entered as an integer or to a level of hundredths of seconds using the n.nn syntax. If a 0 is provided as the CPUTIME for a thread sub-group, there will be no CPU limit placed upon programs running in the associated threads.
real	Default: 3.00 Seconds Specifies the elapse time in seconds for the thread sub-group after which, a message will be issued to the console if the user program has not given up control of it's thread. This value may be entered as an integer or to a level of hundredths of seconds using the n.nn syntax. If 0 is specified, no elapsed time checking will be done for the thread sub-group.
key	Default: M This defines the key in which the threads within the sub-groups will be allocated. If 'M' is specified, the thread keys will be a mixture of user keys excluding the key in which Com-plete is running. If 'N' is specified, no storage protection will be implemented and all threads will run in the same key as Com-plete. The user may also explicitly specify a value in the range 1 to 15 inclusive which will cause each thread to be allocated that key explicitly.

Notes:

A maximum of 8 thread groups may be defined.

A maximum of eight sub-groups can be allocated per thread group. The thread sub-groups may be defined on one line or may be defined on different lines. When a second THREAD-GROUP statement is used, the same group name must be specified to relate the sub-group entries.

Thread group and sub-group names will be upper cased prior to being processed, therefore, entering the parameter in lower case will be treated as, and appear in, upper case letters.

Where more than one specification appears for a thread sub-group for a thread group, the last valid specification will be used.

The amount of storage specified on the THSIZEABOVE System Parameter will be allocated above the line for each thread defined as a result of the THREAD-GROUP System Parameter.

The thread group DEFAULT must always exist in the system. If it is not explicitly defined by the installation, the thread group will be built by the system with the default values. If it is defined, the system insures that a thread sub-group with a thread size at least as large as that required by default is allocated. If not, the system will allocate an additional sub-group for the group. If too many sub-groups have been defined, the last one defined will be overwritten to allow for the default specification.

The keyword data is processed from left to right. If more than one thread sub-group is defined on the one line and the line contains an error, even if an error message is issued for the line, any sub-groups processed up to the error will still have been accepted. That is to say, if the first sub-group is correct and the second is not, an error message will be issued but the first thread sub-group will be defined while the second and subsequent specifications in the same statement will be ignored.

Examples:

```
THREAD-GROUP=(DEFAULT,(SMALUTIL,80,3),(BIGUTIL,300,2,5,9,15))
```

This will cause the DEFAULT thread group to be allocated with two sub-groups. The first sub-group will be called SMALUTIL, will contain three threads with 84K below the line and will take the defaults for CPUTIME, REALTIME and the protectkey to be allocated to the thread. The second sub-group will be called BIGUTIL, will contain two threads with 304K below the line, will have a maximum of CPUTIME of 5 CPU seconds, a REALTIME value of 9 seconds and each thread will have a storage protectkey of 15.

The following sets of System Parameters would cause exactly the same thread sub-groups to be defined:

```
THREAD-GROUP=(DEFAULT,(SMALUTIL,80,3))
THREAD-GROUP=(DEFAULT,(BIGUTIL,300,2,5,9,15))
THREAD-GROUP=(DEFAULT,(SMALUTIL,40,8))
THREAD-GROUP=(DEFAULT,(BIGUTIL,300,2,5,9,15))
THREAD-GROUP=(DEFAULT,(SMALUTIL,80,3))
```

The following sets of System Parameters would cause exactly the same thread sub-groups to be defined in two thread groups, one called DEFAULT and the other called EXTRAGRP

```
THREAD-GROUP=(DEFAULT,(SMALUTIL,80,3))
THREAD-GROUP=(EXTRAGRP,(BIGUTIL,300,2,5,9,15))
THREAD-GROUP=(EXTRAGRP,(SMALUTIL,80,3))
THREAD-GROUP=(DEFAULT,(BIGUTIL,300,2,5,9,15))
```

THSIZEABOVE

Value	n
Default	None

Specifies the amount of storage above the 16 MB line, in multiples of 1024 bytes, to be allocated to each thread.

TIBTAB

Value	name DYNnnnnn ANYnnnnn
Default	TIBTAB=DYN00050

Specifies the name of the terminal definition table to be loaded when Com-plete is initialized. A TIBTAB's load module's RMODE attribute is honored.

TIBTAB=DYN00050 indicates that a TIBTAB is to be built at initialization containing 50 free TIBs.

TIBTAB=ANY01000 indicates that a TIBTAB containing 1000 free TIBs is to be built above the 16 MB line.

TRACECLASS

Value	class (class,OFF)
Default	NONE

Specifies that the indicated class of trace event is to be included in, or excluded from, the Com-plete trace table.

Note that *class* is the name of a valid trace class, as follows:

GENERIC	Used for support purposes
QTIB	TIB queue management
OP	Application program requests
FIXBPOOL	Fixed-length buffer pool operations
VTAM	VTAM operations
ROLL	Roll processing events
ACCESS	ACCESS terminal support
SDFILE	SD File processing
RESOURCE	Resource manager get/free
DISPATCH	Dispatcher events
PRINT	Printout transmission

Note also that (class,OFF) indicates exclusion of the specified class.

TRACEOPTION

Value	option
Default	No options active

Specifies that the indicated *option* is to be either in effect or not in effect for this execution of Com-plete.

Note that *option* is the name of a valid trace option, as follows:

ABEND the trace will continue to run during Com-plete abnormal termination. (Normally the trace will stop recording at the first indication of abnormal termination.)

CAPTURE The trace data will be written to the capture data set. For this option to have any effect, capture processing must be active. Please be aware that this option can cause a huge increase in the activity on and amount of data written to the capture data sets depending on the classes being traced.

EXTENDED Specifies that trace processing is to use the extended form of the trace record. This should only be used where requested by support personnel where specific information must be found as it decreases the amount of trace records that can be held in a trace buffer. Refer to *Main Storage Estimates* for more information.

Note:

TRACEOPTION=ABEND should only be set when requested by Com-plete support personnel.

TRACETABLE

Value	n nK
Default	TRACETABLE=8K

Specifies the size of the Com-plete trace table, which is used to trace events occurring within the Com-plete system.

Note that use of the TRACETABLE sysparm can be a valuable tool for problem resolution.

The minimum size of the trace table is 8K.

TRACETABLE=0 indicates that no tracing is performed.

ULOGM

Value	* password
Default	ULOGM=*

Specifies an override password for access to the Com-plete online maintenance utilities (for user ID maintenance, global PF key maintenance etc.). This should only be used in an emergency when the current password cannot be established. When "*" is specified, as is the normal case, the current password must be used. *password* can be a 1 to 12 character string (excluding commas) which will then become the current password only for the current Com-plete run. The current password should immediately be changed during such a run so that this parameter can be set back to "*".

The default SYSPARM member provided on the distributed source specifies PASSWORD as the current password. You must set a current password online and remove this specification as soon as possible after the installation is finished.

UQDEFAULT

Value	YES NO
Default	UQDEFAULT=YES

Specifies the access restriction option to be exercised by the UQ utility program against terminal users when viewing SYSOUT/SYSLST data sets.

UQDEFAULT=YES specifies that all terminal users are allowed to view all SYSOUT/SYSLST files for any job that does not contain UQ authorization control statements.

UQDEFAULT=NO specifies that no terminal user can view any sysout files for any job unless the job contains proper UQ authorization control statements.

The UQ functions affected by this sysparm are the S, H, C, and R commands and the DE and OC keywords.

For more information concerning UQ authorization control statements, see **UQ - System Job Queue Display Utility** in the Com-plete Utilities documentation.

VSAMBUFFERS

Value	NO (size=n) (sizeK=n) (size=n,sizeK=n,size=n)
Default	VSAMBUFFERS=NO

Specifies the configuration of the VSAM local shared resource buffer pool, SHRPOOL=0.

VSAMBUFFERS=NO specifies that no VSAM LSR pool is to be built.

Here, *size* indicates the size of an individual buffer. Note that size specified without the K option indicates the amount of storage, in bytes, to be reserved; size specified with the K option indicates the amount of storage, in multiples of 1024 bytes, to be reserved. If size is specified without the K option, the permissible values are 512, 1024, 2048, and 4096, or any other number that is a multiple of 4K (4096), up to a maximum of 32K. If size is specified with the K option, the permissible values are 1K, 2K, and 4K, or any number that is a multiple of 4K.

Note that *n* specifies how many buffers are to be allocated. Permissible values for *n* range from 3 to 32000.

VSAMDS

Value	n
Default	4

Specifies the initial number of ACBs in the Com-plete VSAM ACB pool. If necessary during processing, this pool will be expanded dynamically by the same number of ACBs.

n must be numeric.

VSAMFIX

Value	BUFFERS IOBS
Default	VSAMFIX=NONE

Specifies which VSAM areas, if any, are to be page-fixed in virtual storage.

VSAMFIX=BUFFERS and VSAMFIX=IOBS cause the appropriate parts of the VSAM local shared resource pool to be page-fixed. These values have effect only if a local shared resource pool is established using the VSAMBUFFERS sysparm.

VSAMHIPERSPACE

Value	NO (size=n) (sizeK=n) (size=n,sizeK=n,...size=n)
Default	NO

This parameter is valid for MVS ESA systems only.

Specifies the sizes and numbers of buffers in hiperspace to be allocated for the VSAM LSR pool defined by sysparm VSAMBUFFERS. Here, size indicates the size of an individual buffer in bytes (when specified without the K option) or in Kbytes. Permissible values are 4096 / 4K and multiples thereof, up to a maximum of 32768 / 32K.

n specifies the number of buffers of the appropriate size to be allocated.

Note:

If you specify VSAMHIPERSPACE, VSAMBUFFERS must also be specified. Moreover, for each size requested for hiperspace buffers, you must also specify buffers in VSAMBUFFERS.

VSAMRPL

Value	n
Default	The number specified or defaulted for sys-parm VSAMDS multiplied by the number of threads defined.

1. If a VSAM LSR pool is established using the VSAMBUFFERS sysparm, the VSAMRPL value is used for the STRNO parameter of the VSAM BLDVRP macro when building the local shared resource pool, that is, it specifies the maximum number of requests concurrently active for all data sets sharing the LSR pool. If this value is exceeded, VSAM denies additional requests.
2. It specifies the initial number of VSAM RPLs in the Com-plete VSAM RPL pool. If necessary during processing, this pool will be expanded dynamically by the same number of RPLs.

n must be numeric.

VTAMAPPL

Value	name
Default	VTAMAPPL=COMPLETE

Specifies the name that Com-plete is to use as the application ID with VTAM.

For more information concerning VTAM definitions, see *Software Interfaces*.

VTAMBUFFER

Value	n
Default	VTAMBUFFER=3584

Specifies the size of the VTAM RECEIVE ANY buffer.

Note that *n* should be set to a value equal to or greater than the size of the largest incoming data. The minimum size is 128 and the maximum is 32767 bytes. If an incoming RU length exceeds the specified buffer size, the message ZVT4010 is logged and the RU is ignored. If the buffer size is too large, storage is wasted.

Note:

VTAMGENERIC

Value	name
Default	None

Specifies the name used by Com-plete to identify itself to VTAM as a generic resource. It is used in Parallel Sysplex installations to achieve system-independence and workload-balancing. For more details and restrictions of generic resource names refer to the section on VTAM interface in the *ChapterSoftware Interfaces*.

VTAMPASSWORD

Value	password
Default	None

Specifies the password to be used when the VTAM ACB is opened.

VTAMSIMLQ

Value	YES NO
Default	VTAMSIMLQ=YES

Specifies whether or not the SIMLOGON performed for each LU to be acquired is queued. For more information, see the description of the SIMLOGON in the *IBM VTAM Macro Language Reference documentation*.

VTAMSIMLQ=YES indicates that the SIMLOGON done for each acquired LU is to be queued.

VTAMSIMLQ=NO indicates that the SIMLOGON is not to be queued.

VTAMSIMLREL

Value	YES NO
Default	VTAMSIMLREL=YES

Specifies whether or not the SIMLOGON performed for each LU to be acquired will include OPTCD=RELREQ. This OPTCD causes VTAM to ask the current owner (if any) to release the LU.

VTAMSIMLREL=YES indicates that the SIMLOGON for each acquired LU will include OPTCD=RELREQ. If YES is indicated, the VTAMSIMLO startup option must be YES.

VTAMSIMLREL=NO indicates that the SIMLOGON for each acquired LU will not include the OPTCD=RELREQ.

VTAMSTART

Value	YES NO
Default	VTAMSTART=YES

Specifies whether or not VTAM processing is to begin when Com-plete is started. (VTAM can be stopped or started using the Com-plete operator commands VTAM START and VTAM STOP.)

VTAMSTART=YES indicates that VTAM processing will begin when Com-plete is started.

VTAMSTART=NO indicates that VTAM processing will not begin when Com-plete is started.

WTOBUFFERS

Value	n (n,BELOW) (n,ANY)
Default	WTOBUFFERS=0

Specifies the number of messages written by Com-plete to the operators console that are to be retained in storage. These messages can be displayed by the MO function of UUTIL (subfunction CM).

Note:

each buffer requires approximately 140 bytes of storage.

buffers will be allocated above the line for XA and later systems. To allocate buffers below the line, use the notation (n,BELOW).

Binary Modifications (APPLYMODS)

The following is a list of the system modifications available for Com-plete. The modifications are implemented using the APPLYMOD parameter during Com-plete initialization.

Most applymods can be changed online by control users who have entered the correct system maintenance password for the UUTIL utility.

1	When a hardcopy request is issued from Com-plete either via the "*UCOPY" command or the hardcopy program function key, a page eject will normally occur BEFORE each hardcopy. Specify APPLYMOD=1 to avoid the page eject BEFORE each hardcopy.
2	When a hardcopy request is issued from Com-plete either via the "*UCOPY" command or the hardcopy program function key, a page eject will normally NOT occur AFTER each hardcopy. Specify APPLYMOD=2 to force a page eject AFTER each hardcopy.
3	Com-plete calls the user exit ULSRPSFS to perform security validation each time an application program requests a Com-plete function. Specify APPLYMOD=3 if your installation wants Com-plete to call ULSRPSFS for each internal Com-plete operation and for any SVC that Com-plete traps. Note: This modification has serious performance implications. Before installing this modification, consult your Software AG technical representative.
4	The Com-plete utility UEDIT will delete its SD file when the terminal operator terminates a UEDIT session. Specify APPLYMOD=4 to instruct UEDIT not to delete its SD file during termination.
5	By default, Com-plete transmits the hello message to the terminals on the network without the date or time of day displayed. Specify APPLYMOD=5 to force Com-plete to display the date and time on the hello message.
6	The *UQ utility and the MO function of *UUTIL only allow control terminals to issue operator commands. Specify APPLYMOD=6 to allow non-control users to issue operator commands.
7	The *UM utility only allows control terminals to issue the TID functions. Specify APPLYMOD=7 to allow non-control terminals to issue the TID functions.
8	By default, unused parts of a thread are always cleared when a new program is being started or rolled in. This is for security reasons, to ensure no data is left behind from the application that last used the thread. When your security policy allows it, you can gain slightly better performance with this applymod set.

9	By default, only those parts of a thread are dumped which are actually in use by the application running. This applymod can help determine problems caused by a destroyed free area chain. Running with this applymod set causes bigger thread dump sizes and, consequently, possibly fewer dumps fitting into the dump dataset.
10	By default, UQ uses its own security mechanism. Set this Applymod to make UQ use the definitions in the SAF classes JESJOBS and JESSPOOL instead.
11	The *ULOG utility will suppress display of the account number on the logon screen (ULGO) when a user logs on to Com-plete. Specify APPLYMOD=11 to force display of the account number.
12	The *UQ utility passes the first "//*UQ user ..." card image to the user exit UUQEX1. Specify APPLYMOD=12 to force the *UQ utility to pass the first of any "//*UQ ..." cards (except the //*UQ ALLOW card) to UUQEX1.
13	<p>The Com-plete mapping system will not rewrite the constant fields of a map to the terminal if the screen has been changed by non-mapping write requests. Specify APPLYMOD=13 to cause the mapping system to always rewrite the constant fields of a map.</p> <p>Note: There is a mapping TCC code to force this same result for a particular map. It is advisable to use the new TCC code rather than this modification.</p>
14	Com-plete does not allow batch programs to send priority messages (class=2) to terminal users. Specify APPLYMOD=14 to allow priority messages to be sent from batch programs.
15	If Com-plete gets an I/O error while trying to send a message or a printout to a terminal, Com-plete will put the current MSG/PO in hold and continue with the next MSG/PO in the queue. Specify APPLYMOD=15 to prevent transmission of MSG/POs to a terminal if a MSG/PO gets an I/O error. For Com-plete to retry sending the queued messages, use the *UM utility to reset the terminal.
16	The UEDIT utility saves the current member when the SUBMIT command is entered. Specify APPLYMOD=16 to change the default to NOSAVE. Then, to save the current member, specify SAVE with the SUBMIT command.
17	The Com-plete utility *UEDIT fetches *UQ after each SUBMIT command. Specify APPLYMOD=17 to force *UEDIT not to fetch *UQ when a SUBMIT command is issued.
18	Currently no function is assigned to this applymod.
19	Currently no function is assigned to this applymod.

20	If an application program does an MCALL WRTS for a TTY device, Com-plete will not append carriage return and line feed control characters to the output message. Specify APPLYMOD=20 for Com-plete to append carriage return and line feed control characters to the output message.
21	This modification applies to TTY devices only. Installations that have an auto speed front-end require that the mainframe issue a PREPARE CCW (instead of a WRITE CCW) immediately following the ENABLE CCW. Specify APPLYMOD=21 to issue a PREPARE CCW immediately following the ENABLE CCW.
22/23	This modification applies to TTY devices only. Specify APPLYMOD=22 to not write the characters "-BREAK-" to a TTY whenever output is terminated via the break or interrupt key. Specify APPLYMOD=23 to write the characters -BREAK- to a TTY when input is terminated via the break or interrupt key.
24	The ULOG utility allows a logged-on user to change the password currently assigned to that user. Specify APPLYMOD=24 if the user is not to be allowed to change the password. Note: If APPLYMOD=24 is specified, the NEWPASSWORD parameter for ULOG is invalid.
25	This modification influences the format of the date returned by the DATE / DATEJ API functions. APPLYMOD=25 causes a century indication to be returned: Function//without Applymod 25//with Applymod 25 DATE//X'0mmddyF'//X'cmmddyF' DATEJ//X'00yydddF'//X'0cyydddF' where 'c' is the century minus 19 (0 for 19, 1 for 20, ...). See the Application Programmer's Manual for more details.
26	This modification applies to TTY devices only. When the break key is pressed, Com-plete will receive input from a TTY device. Specify APPLYMOD=26 to cause Com-plete to ignore input from a TTY when terminated via the break key.
27	Com-plete normally handles a session with any device as one bracket. Specify applymod 27 to allow an application to request sending of End Bracket and FM-Headers.
28	Set ATTN as SUSPENDKEY.
29	For VTAM terminals only to set the line length and the number of lines according to the PSERVICE values instead of using the default Model 2.
30	Some TTY interface hardware/software requires that the output data stream end in an ETX character (X'03'). Specify APPLYMOD=30 to have Com-plete add the ETX character to the output data stream.

31	The user-written routine ULSRPSFS is called twice during program "FETCH" functions. Specify APPLYMOD=31 to prevent the 0 (program initialization) call to ULSRPSFS during FETCH function processing and only pass the 12 (FETCH) call.
32	APPLYMOD=30 adds an ETX to the output buffer for all types of writes to TTY devices. Specify APPLYMOD=32 to add the ETX only on WRTC and WRTD writes.
33	Some TTY interface hardware/software does not support the DC1 character (x'11'). Specify APPLYMOD=33 to not add a DC1 to the output data stream.
34	The regular hello message can fill up small buffers on large networks. Specify APPLYMOD=34 to send the short hello message (normally sent to TTYs and printers) to all terminals.
35	By default, UPCBs (user program control blocks) are located above the 16 MB line in this version. Use Applymod 35 to build UPCBs below the 16 MB if this is required. Note that this Applymod cannot be changed for the running Com-plete.
36	TLAMTTY issues halt I/O for "prepare channel program". Specify APPLYMOD=36 to eliminate halt I/O.
37	Currently no function is assigned to this applymod.
38	<p>Com-plete normally forces an "End Bracket" after each printout to allow the controller to insert local hardcopies. However, some controllers simply ignore the "EB", causing the session to stay "In Brackets". The next "BB" is then rejected with SENSE 0813 (Bracket Bid Reject). Specify applymod 38 to suppress the "EB" at end of printouts. This will cause all queued printouts to be sent in one Bracket.</p> <p>Important: Static Printers should always be defined with OPT=SHARE to force a CLSDST after the last printout.</p>
39	During printing of a spooled printout, Com-plete records the number of printout copies in storage. If Com-plete is terminated before completion of all copies of the printout, the printout will be restarted and all copies will be printed again. Specify APPLYMOD=39 to force Com-plete to decrement the number of copies in the restart information.
40	COM-PASS users receive a ULOG menu screen when ENTER is pressed and no user is logged onto the TID. Specify APPLYMOD=40 to eliminate this screen.
41	Class 2 messages destroy the current screen display. This results in the loss of all screen updates since the last pressing of the key. Specify APPLYMOD=41 to use class 5 as the urgent message class. Messages will then be sent after the key is pressed and all screen data are read. User IDs and TIBTAB entries must be changed to receive class 5 messages. Class 2 message support remains unchanged.

42	Hold printouts not closed if a program abends. When a program terminates under Com-plete either normally or abnormally, any open printouts are closed and scheduled. If this applymod is on, when a program terminates abnormally, the open printouts will be closed, however, they will be flagged as "held". It will then be up to you to documentionly release the printout to be printed or to purge it.
43	Normally you can send messages from application programs using MESGSW to any user, irrespective if he is logged on or not. Specify APPLYMOD=43, if you want to send messages only to users who are currently logged on.
44	UQ S assumes that printouts in the JES spool created without a RECFM parameter in the DCB contain an ASA carriage control character in each first column. Specify APPLYMOD=44, if the output contains only data and you want Com-plete to insert an additional blank into each row, when such a printout is transferred with UQ S DC= into the Com-plete spool data set.
45	Normally the logical output driver is deleted from storage, when the usecount gets zero. Specify APPLYMOD=45 to cause the driver module to stay in storage.
46	When figures are being displayed, "." is used to denote a decimal point and "," is used to punctuate integers e.g. 100,300.54 is one hundred thousand three hundred point fifty four. In some countries, the use of "," and "." is reversed. Specify APPLYMOD=46 to cause figures to be displayed in this reversed way (the above example appears as 100.300,54 with applymod 46 on).
47	Currently no function is assigned to this applymod.
48	Whenever possible, Com-plete uses Exception Response to communicate with Terminals and Printers. However, some printers - especially when connected to a LAN - might lose parts of the output when using this protocol and require Definite Response. Specify Applymod 48 to force use of Definite Response for Printers.
49	Currently no function is assigned to this applymod.
50	Messages ZUS0001 through ZUS0004 are written to the console when the USTOR AM function is invoked. Specify APPLYMOD=50 to inhibit the writing of these messages to the operator console.
51	Normally the record format for a BDAM file is hardcoded to be fixed. Specify APPLYMOD=51 to take the record format from the JCL specification.
52	Com-plete ignores any data passed on the VTAM logon. Specify APPLYMOD=52 to have the VTAM interface interpret the data and initiate a logon. Please refer to the VTAM section of the chapter Software Interfaces for more information.

53	At Com-plete logoff from a VTAM terminal, the terminal will be disconnected from Com-plete and returned to VTAM. Specify APPLYMOD=53 to cause VTAM terminals to remain connected to Com-plete after logoff. A new logon will still be required.
54	At Com-plete logoff from a dialup terminal, the terminal will be disconnected from Com-plete and the line will be "hung up". Specify APPLYMOD=54 to cause dialup terminals to remain connected to Com-plete after logoff. A new logon will still be required.
55	Com-plete will accept logon from any VTAM terminal. Specify APPLYMOD=55 to cause Com-plete to only accept logon from VTAM terminals that are defined in Com-plete's TIBTAB.
56	Currently no function is assigned to this applymod.
57	Com-plete only accepts logons from user IDs that are defined to Com-plete or, if the Natural Security interface has been installed, if the user ID is defined to Natural Security. Specify APPLYMOD=57 to accept logon from any user ID not defined to Com-plete or Natural Security. The password entered is ignored unless an external security system is active. Logon is then subject to the external security system. See the chapter Software Interfaces for more information.
58	Currently no function is assigned to this applymod.
59	Normally, Com-plete maps display in both upper and lowercase. Specify APPLYMOD=59 to cause all constant text fields in Com-plete system maps to be converted to uppercase.
60	Normally, Com-plete will scan the LPA or SVA in order to find a module. Specify APPLYMOD=60 to not scan the LPA or SVA on Com-plete COLINK's and COLOAD's (in order to reduce overhead for performance reasons).
61	Normally, Com-plete system maps display the date in American format, e.g., MM/DD/YY. Specify APPLYMOD=61 to cause Com-plete system maps to display the date in European format, e.g., DD.MM.YY.
62	Currently no function is assigned to this applymod.
63	Currently no function is assigned to this applymod.
64	Currently no function is assigned to this applymod.
65	The mapping subsystem of Com-plete normally only accepts positive numbers in a field defined to be zoned in the map. Specify APPLYMOD=65 to force the mapping subsystem to accept negative values in zoned fields on maps.
66	When a user program does not request that to should be indicated to the program, Com-plete assigns default Com-plete paging commands to them. Specify APPLYMOD=66 to force Com-plete not to assign defaults to the PF keys when the user program does not request that they be returned.

67	The UQ function to display a copy of what is on the console (that is, UQ M) normally displays the outstanding WTOR requests in the first lines of the screen followed by the WTOs. Specify APPLYMOD=67 to avoid the display of the outstanding WTORS at the start of the UQ M screen.
68	For the UQ print out spooling functions "PT" and "DC", the output is not translated. Specify APPLYMOD=68 to force the output created by these two commands to be translated.
69	If APPLYMOD=29 is specified and therefore Com-plete is taking the screen size from the VTAM bind image, the terminal will by default be set to use the alternate screen sizes specified in the VTAM bind image. Specify APPLYMOD=69 to force Com-plete to set the terminal to use the primary screen sizes specified in the VTAM bind image. Note that if the terminal has been defined in the TIBTAB, the setting of OPT=ALTE NOALTE will override this applymod.
70	For a hardcopy request, Com-plete simply prints a hardcopy of the screen currently displayed at the user's terminal. Specify APPLYMOD=70 to cause the hardcopy to be printed with header information containing the initiating user ID, the TIB name, the TIB number, the installation ID of the Com-plete, the date and the time of the hardcopy request.
71	For a printout spool request, Com-plete simply prints the first page of the printout according to the control character specified when the printout was created. Specify APPLYMOD=71 to get Com-plete to force a form feed at the start of every printout printed. Please note that when applymods 71 and 87 are specified, applymod 87 takes precedence. Note that a hardcopy request is also seen as a "printout" and is therefore also affected by this applymod.
72	For a printout spool request, when a printout ends, Com-plete starts the next printout directly afterwards. If there is no form feed in the following printout, this printout will directly follow the previous printout on the paper. Specify APPLYMOD=72 to get Com-plete to force a form feed at the end of every printout printed. Please note, a hardcopy request is also seen as a "printout" and will therefore also be effected by this applymod.
73	When an abend occurs in Com-plete, recovery will be performed by Com-plete and the application will receive control back if abend exits have been set or the application program will be abended if not. Specify APPLYMOD=73 to force Com-plete to request an operating system dump prior to recovery. This will cause the standard ABE messages to be printed and a dump to be taken according to the installation dump options set for the operating system. Please note, this applymod is required to produce additional diagnostics in a situation where a thread dump does not suffice. The installation could suffer severe performance problems if this applymod is set for any length of time and therefore it should only be set at the request of your support representative.

74	Currently no function is assigned to this applymod.
75	When an abend occurs in Com-plete for a system task (e.g. TAM), or when a thread abend occurs within recovery processing for a user program, Com-plete writes various ABE messages to the console, takes an operating system dump, and then attempts recovery. Specify APPLYMOD=75 to force Com-plete to simply issue a minimum number of ABE messages to the console and attempt recovery without taking an operating system dump. Please note, this applymod is available to assist in a situation where abends are seriously impacting the system performance while the problem is being investigated. It should only be set on the advice of support personnel as any problems that may occur with this applymod on cannot be investigated as there will be no diagnostic information available.
76	Currently no function is assigned to this applymod.
77	For a user request, the parameters passed are normally verified as to whether the areas exist and as to their location. Specify APPLYMOD=77 to avoid this checking. Com-plete then assumes that all areas are correct. This applymod is intended as a performance enhancement for a production system where things change very infrequently. You are recommended to set this applymod only in very stable environments, and that it can easily be turned off if problems arise.
78	With this applymod, Com-plete will load a reentrant load module requested to be executed or loaded as part of an application as if it was specified using the RESIDENTPAGE sysparm.
79	Programs ATTACHed or FETCHed in a thread (including initial program invocation) must normally be cataloged by ULIB only if special parameters such as region size, thread lock are required. Programs that do not require such parameters or use values defined by \$DEFAULT need not be cataloged. Specify APPLYMOD=79 to force Com-plete to ATTACH and FETCH only programs cataloged by ULIB. Programs not cataloged cause return codes or error messages to be issued as if they did not exist.
80	Normally, a program must be cataloged by ULIB only if it is intended to be initially invoked in a thread (ATTACH/FETCH), and therefore requires special parameters such as region size and thread lock. Programs loaded by other programs only, and programs that need no special parameters or use values defined by \$DEFAULT need not be cataloged. Specify APPLYMOD=80 to force Com-plete to load only programs cataloged by ULIB. Programs not cataloged cause return codes or error messages to be issued as if the program did not exist. Note that the loading of maps is not affected by this applymod. When APPLYMOD=80 is specified, you must first run a TULIB batch job using CATAPM80 from the Com-plete source library as input, otherwise Com-plete will not function properly

81	When a request is made against a System Data Container, SDAM searches for "global data only" by default.. Specify APPLYMOD=81 to force SDAM to search for a "local" copy (that is, the data relating to the current patch character) before using the global copy. Note that in a worst-case scenario, this may nearly double access rates against the System Data Container(s).
82	Setting this Applymod helps avoid ROL0009 errors which occur under certain circumstances when running Natural 2.2 and above.
83	If an external security system is specified via the SECSYS Com-plete sysparm, when a user logs on, the password entered is encrypted and saved in a Com-plete control block. It is encrypted in such a way that it can be decyphered again if the "actual" password is required for any reason, for example, job submission in a RACF environment. Specify APPLYMOD=83 to force Com-plete not to save the password in a Com-plete control block that is, Com-plete does not save the password anywhere. Please note that due to early verify techniques for job submission from TOP SECRET and ACF2, this applymod can be specified for these two external security systems. However, this is not possible with RACF, as it can lead to job submission errors.
84	When a print request is issued via the Com-plete utility program U2PRINT as is the case with all Com-plete utilities and Natural, a map is presented with the selected options which can then be changed by the user. Specify APPLYMOD=84 to avoid the display of this map.U2PRINT simply prints the data using the values initially supplied.
85	When a printer is allocated dynamically via the key in U2PRINT, it is allocated with no alternate device and as an active printer. Specify APPLYMOD=85 to cause dynamic printers to be allocated with the SYSOUT printer as the alternate device and DEL=YES.
86	<p>When Com-plete comes up normally, it uses a standard operating system mechanism to set the address space non-swappable for performance reasons. Specify APPLYMOD=86 to avoid this call so that Com-plete is left as a swappable address space. This applymod is primarily designed for test or other systems that should not impact the general performance of the machine. If ACCESS is specified, this applymod has no effect because Com-plete becomes a cross-memory service as soon as it signs on to the Adabas router; to be a cross-memory service, the address space must be non-swappable.</p> <p>Note: This applymod is valid for MVS systems only.</p>

87	For a printout spool request, Com-plete simply prints the first page of the printout according to the control character specified when the printout was created. Specify APPLYMOD=87 to get Com-plete to force NO form feed at the start of every printout IF the printout specifies that a form feed should occur. This means that if the control character in column 1 is "1", it will be changed to blank if this applymod is on. Note that a hardcopy request is also seen as a "printout" and will therefore also be effected by this applymod.
88	Currently no function is assigned to this applymod.
89	When external security checking is active, informational messages can be returned by the external security during logon. Specify this applymod to cause these messages to be displayed at the user terminal before the Com-plete message ULG0003 Logon Successful .
90	Normally VTAM terminal line-/page-size are extracted from the BIND image. Specify Applymod 90 to force the extraction of the terminal's ls/ps from the WSF QUERY reply instead of the BIND image.
91	Use the OS SNAP function to take a dump When an error occurs within Com-plete other than an application program error, a dump will normally be scheduled by Com-plete's recovery processing. This will cause a dump to be written to the SYSUDUMP, SYSABEND or SYSMDUMP DD statement as per normal OS rules. When this applymod is specified, an output dataset will be allocated and opened and the OS SNAP function will be used to take a dump to that DD statement. This is a very slow method of taking a dump and it is recommended that this applymod NOT be set for MVS systems. It is designed for FACOM systems where either this applymod or applymod 92 must be specified and where the installation feels that it cannot accept the delay applymod 92 may impose on the system.
92	Use the OS SDUMP function to take a dump When an error occurs within Com-plete other than an application program error, a dump will normally be scheduled by Com-plete's recovery processing. This will cause a dump to be written to the SYSUDUMP, SYSABEND or SYSMDUMP DD statement as per normal OS rules. When this applymod is specified, Com-plete will use the OS SDUMP function to take a dump. This will result in a unformatted dump being written to the SYS1.DUMP dataset(s) which can be formatted at some later stage. This method of taking a dump is as quick as taking a dump to the SYSMDUMP DD statement, however, it will cause the entire MVS system to stop for a number of seconds while certain control blocks are copied. It is recommended that this option be set for FACOM systems as it takes a more comprehensive dump and will be faster than taking a SNAP dump which should compensate for the minor system hiccup which it will produce.
93	Translate all messages to upper case.
94	Translate FACOM SO/SI to IBM SO/SI in DBCS datastream.

95	Always append a "new line" character at the end of a printout. This serves printers that require the new line to reset the print head to the correct position.
96	Always send the message ULG0007 Logoff Successful after logoff. Com-plete normally only sends this message to terminals which require that it be sent such as BSC terminals or ACCESS terminals. It is normally not required for SNA terminals as they will normally be presented with the Unformatted System Services map when the VTAM CLSDST is issued. There are VTAM SNA devices however, that have a dependency on this message being sent due to procedures being run there by Entire Connection or other emulation tools. Setting this applymod will ensure that the message is always sent to the terminal.
97	Ignore any forms specifications for printouts with a logical output driver specified. When a printout is printed with a form specified, the printout will not be printed by Com-plete until the same form is mounted at the printer. Many of today's intelligent printers can have forms mounted via escape sequences or the form may be "flashed" by the printer itself. As many users would use a logical output driver for this purpose, this applymod will insure that users can specify the form in the appropriate field in the application, Com-plete will schedule the printout regardless of the logical form mounted according to Com-plete, and the logical output driver can cause this form to be "mounted" on the printer.
98	Always allocate the UAB/STCK control block below the 16 MB line. Com-plete will allocate any User Accounting Blocks and Stack Control blocks above the line normally. Some user exits and applications will not be able to handle this, therefore, specifying this applymod will force Com-plete to get these control blocks below the 16 MB line. This applymod is being provided to assist users in migrating their exits and applications to 4.6. It is recommended that all exits and applications are changed to support the control blocks above the line as this applymod will disappear at some point in the future.
99	This applymod is related to the printout menu which apperas when you invoke a print function from some Com-plete utilities or from Natural. By default, if the printer you specify is not defined in the TIBTAB, you have to press PF5 to have the printer allocated dynamically. With this applymod, pressing ENTER has the same effect as PF5.
100	This applymod is related to the Printout Menu which appears when you invoke a PRINT function from some Com-plete utilities or from Natural. By default, you can enter a printer name or a TIB number into the destination field. If your site uses only dynamic printers, specifying TIB numbers makes no sense. It even can cause printouts to be misrouted to the wrong printer or to someone's terminal. With this applymod set, Com-plete rejects numeric printout destinations.

Defining Terminals and Printers

This chapter describes Com-plete's Terminal Information Block Table (TIBTAB).

In the vast majority of environments it is preferable to have Com-plete build the TIBTAB dynamically during startup. All required information about terminals and printers connected using the VTAM access method can be obtained by Com-plete directly from VTAM during LOGON / SIMLOGON.

TCP/IP based connections (Telnet tn3270 sessions and connections to LPD print servers) are supported using dynamically allocated TIBs only, not for TIBs hard-coded in a TIBTAB.

The information in this chapter applies to those installations only which still must maintain and assemble a coded TIBTAB for whatever reason.

This chapter covers the following topics:

- Overview
 - How to Code TIBTAB
 - How to Create TIBTAB
 - Printout Spooling TIBTAB Considerations
 - Batch Output Facility (SYSOUT)
 - Logical Output Drivers
 - Line Printer Daemon (LPD) Protocol
-

Overview

The Terminal Information Block Table (TIBTAB) defines the terminals and lines to be used by Com-plete. TIBTAB resides as a load module in the load library, and is loaded by the Com-plete control program during initialization.

Since the TIBTAB exists as a member of the load library/core image, the maximum number of characters that can be used to make up the name is eight. Multiple copies under separate names can exist.

When you create a new TIBTAB, it is recommended that the module name used be different from the module name of the current working TIBTAB. If the two module names are the same and an undetected error exists in the newly created TIBTAB, Com-plete initialization will fail.

How to Code TIBTAB

The TIBTAB is coded as an Assembler language module using the following macros and compiler instructions:

- TIBSTART
- TIB
- Translation Table Definitions
- CMDEVS
- TIBEND
- END

These instructions are combined to create a non-executable module, which is nothing more than a table.

A sample TIBTAB is supplied in the source library (member TIBTAB).

TIBSTART Macro

The TIBSTART macro defines the number of Terminal Information Block (TIB) entries to be generated.

A TIB entry is required for each terminal in the network. In addition, a TIB entry is required for each simultaneously-executing batch job that uses a Com-plete function, as well as for each simultaneously-executing online program invoked by the ATTACH function of Com-plete.

The format for using the TIBSTART macro is:

```
name TIBSTART NOTIBS=n
  [ ,SMC=(c1,c2,...,cn) ]
  [ ,RMC=(c1,c2,...,cn) ]
  [ ,VAL={YES|NO} ]
```

The arguments are:

name

Required.

Specifies the TIBTAB name.

NOTIBS=n

Required.

Specifies the number of TIB entries to be generated. This number also specifies the largest number allowed when specifying a TID number (TIB macro).

Note that the value specified for NOTIBS must be at least as large as the total number of TIB macros defined in TIBTAB and all the dynamically-allocated ACCESS terminals in the network. If no batch TIBS are specifically defined, then NOTIBS should also be large enough to allow one or more undefined TIB entries to exist.

SMC=(c1,c2,...,cn)

Optional.

Default	SMC=(1)
---------	---------

Specifies the default authorization-sending message class codes to be assigned to all defined printer TIBs without an SMC argument assigned. Authorization message class codes are fully described in the Com-plete Utilities documentation.

RMC=(c1,c2,...,cn)

Optional.

Default	RMC=(1,2,4)
---------	-------------

Specifies the default authorization receiving message class codes to be assigned to all defined printer TIBs without an RMC argument assigned. Authorization message class codes are fully described in the Com-plete Utilities documentation.

VAL=[YES|NO]

Optional.

Default	VAL=YES
---------	---------

Specifies whether or not the TIB parameters are validated. Note that assembly of the TIBTAB is faster if validation of the parameters is bypassed. Care must be taken, however, to ensure that the parameters are correct.

VAL=YES indicates that the parameters are validated.

VAL=NO indicates that the parameters are not validated.

TIB Macro

The TIB macro defines a terminal and its characteristics.

Batch jobs and attached online programs that use Com-plete functions also require use of a TIB entry. TIB entries for batch jobs and attached online programs can be defined explicitly by use of the batch device specification defined below, or are defaulted by specifying a NOTIBS value that is larger than the total number of TIB and LGCB macros defined.

The format for using the TIB macro is:

```
[name] TIB tid,{VTAM|ACCESS},device
[,ALTTID=tid]
[,CTLTIB={YES|NO}]
[,DEL={YES|NO}]
[,FORMAT=n]
[,GROUP=ALL=NO]
[,HC={YES|NO}]
[,LEN=n]
[,LINES=n]
[,LOGOFF={YES|NO}]
```

```
[ ,LOWER={YES|NO} ]
[ ,MAXRU= ]
[ NAME=name ]
[ ,NODEID=node ]
[ ,OPT= ( {ALTE|NOALTE}
, {PAD|NOPAD}
, {MOD3|NOMOD3}
, {EBCDIC|BCDIC|EBCDIC,ADD|BCDIC,ADD}
, {CR|NOCR}
, {EM3270|NOEM3270}
, {SHARE|NOSHARE}
, {ACQUIRE|NOACQUIRE}
, {COMP|NOCOMP}
{SCS|NOSCS}
{IBM|FACOM|HITACHI} ) ]
[ ,PRI=x0|1|2|3|z ]
[ ,RMC=(c1,c2,...,cn) ]
[ ,SCHC=tid ]
[ ,SMC=(c1,c2,...,cn) ]
[ ,STALL={YES|NO} ]
[ ,TRTABS=trantab ]
[ ,VAL={YES|NO} ]
```

The arguments are:

name

Optional.

Specifies the TIB name; for assembly listing purposes only.

The name can be any character string that is valid as an Assembler label.

tid

Required.

Specifies the Terminal Identification number of the terminal being defined. This value must be an integer from 1 to 9999, and must be less than the value for NOTIBS as specified in the TIBSTART macro.

VTAM or ACCESS

Required.

Specifies the access method used for the device.

device

Required.

Specifies the device type of the terminal being defined. This value must be one of the values defined in the terminal device type table in Appendix.

If the NOTIBS argument of the TIBSTART macro specifies a value larger than the total number of TIB, the excess TIBs are reserved for programs invoked by the ATTACH function, and VTAM terminals.

ALTTID=tid

Optional.

Default	None
---------	------

Specifies the TID number of the terminal to receive messages when the terminal being defined is inoperative, or when more than 10 messages are queued for the terminal being defined.

CTLTIB={YES|NO}

Optional.

Default	CTLTIB=NO
---------	-----------

Specifies whether or not the terminal being defined is to be assigned control status. Control terminals have privileged status and are allowed to execute Com-plete utility programs and certain functions reserved for the control user.

Note that at least one terminal must be defined as having control status. This should be the terminal with a TID value of one.

CTLTIB=YES indicates that the terminal being defined has control status.

CTLTIB=NO indicates that the terminal being defined has non-control status.

DEL={YES|NO}

Optional.

Default	DEL=NO
---------	--------

Specifies whether the terminal being defined is to be brought up deleted when Com-plete is initialized.

DEL=NO specifies that the terminal is to be available for use when Com-plete is initialized.

Since Com-plete allows any VTAM terminal to log on regardless of whether it is in the TIBTAB or not, in order to disallow logon from a VTAM terminal, that terminal must be included in the TIBTAB with DEL=YES specified.

FORMAT=n

Optional.

Default	Depends upon the device type as specified in Terminal Device Type Codes
---------	---

Specifies the format to be used for CRT devices.

Here, *n* must be specified as one of the following:

- 240(6x40)
- 480(12x40)
- 960(12x80)
- 1920(24x80)
- 2560(32x80)
- 3440(43x80)
- 3564(27x132)

The arguments `LINES` and `LEN` are automatically set by the value given for the `FORMAT` argument as described by the numbers in the parentheses.

Note that if the `FORMAT` argument is specified in a `TIB`, `LINES` and `LEN` cannot be specified.

GROUP=ALL=NO

Optional.

Default	The associated TID will reside in the <code>ALL</code> grouping.
---------	--

`ALL=NO` indicates that this TID will not be considered part of the `ALL` grouping by message switching. This includes `CALL MESSAGESW`, the hello message, and `*UM`. The `ALL` grouping can also be used in operator commands.

HC={YES|NO}

Optional.

Default:

`HC=YES` for all devices except 3270-type terminals; `HC=NO` for 3270-type terminals.

Specifies whether or not the terminal being defined is a hardcopy device.

`HC=YES` indicates that the terminal is a hard copy device. `NO` indicates that the terminal is not a hardcopy device.

LEN=*n*

Optional.

Default	The minimum and maximum values will depend upon the device type.
---------	--

Specifies the length of the line to be used.

Note:

Do not specify if FORMAT is specified.
Here, n must specify a decimal value.

LINES=n

Optional.

Default	The minimum and maximum values will depend upon the device type.
---------	--

Specifies the maximum number of lines to be used in a display or output.

Note:

Do not specify if FORMAT is specified.
Here, n must specify a decimal value.

LOGOFF={YES|NO}

Optional.

Default	LOGOFF=YES
---------	------------

Specifies whether or not this terminal will be logged off after the period of inactivity defined by the AUTOLOGOFF sysparm.

LOGOFF=YES indicates that the terminal will be logged off for inactivity.

LOGOFF=NO indicates that the terminal will not be logged off for inactivity.

LOWER={YES|NO}

Optional.

Default	LOWER=NO
---------	----------

Specifies whether or not lower-case characters will be accepted from the device as input.

LOWER=YES indicates that all characters will be accepted in lower-case format.

LOWER=NO indicates that only upper-case characters are supported. For input, lower-case characters are translated to upper-case.

MAXRU=n

Optional.

Default	None
---------	------

Specifies an override RUSIZE for a VTAM device.

NAME=name

Optional.

Default	CL8'TIB0nnnn' where nnnn is the four-digit TID number padded with leading zeros if necessary.
---------	---

Specifies a one- to eight-character name to be associated with this TIB. If VTAM is the access method being used, the name must specify the name of the VTAM node (LU) to which Com-plete is to connect (see the ACQ keyword).

Here, *name* is also used in group name searches. TIB name entries will be searched before searching for a group name. The first TIB name will satisfy the search, and the TID of the TIB with that name will be used. If duplicate TIB names are defined, only the first will be found. If a duplicate TIB name and group name are defined, only the first TIB name will be found.

NODEID=node

Optional.

Default	None
---------	------

Specifies the ACCESS node number. This value must be an integer between 1 and 65536. Valid only if ACCESS is specified for the CUU parameter.

Specifies the optional features to be assigned to a terminal.

OPT=value

Format (default is underlined>):

```
OPT=( {MOD3|NOMOD3}, {PAD|NOPAD}, {ALTE|NOALTE},
      {EBCDIC|BCDIC|EBCDIC,ADD|BCDIC,ADD}, {CR|NOCR},
      {EM3270,NOEM3270}, {SHARE|NOSHARE}, {ACQUIRE|NOACQUIRE},
      {COMP|NOCOMP}, {SCS|NOSCS}, {IBM|FACOM|HITACHI} )
```

Meaning of the values:

MOD3|NOMOD3

Optional.

Default	<u>OPT=NOMOD3</u>
---------	-------------------

Specifies whether or not the IBM 3275 terminal being defined has an IBM 3284 model 3 printer attached.

OPT=MOD3 indicates that the IBM 3275 terminal has an IBM 3284 model 3 printer attached.

OPT=NOMOD3 indicates that the IBM 3275 terminal does not have an IBM 3284 model 3 printer attached.

PAD|NOPAD

Optional.

Note:

This operand is ignored if the terminal being defined is not a TTY device.

Default	OPT=NOPAD
---------	-----------

Specifies whether padding (idle) characters should be added to the line protocol output in order to allow for mechanical motion (e.g., carriage return). Because most terminals have internal buffering to account for mechanical motion so that padding is not required, OPT=NOPAD is normally specified.

OPT=PAD indicates that idle characters should be added to the line protocol.

OPT=NOPAD indicates that no idle characters will be added.

ALTE|NOALTE

Optional.

Default	If LINES=24 and LEN=80 or LINES=12 and LEN=40 are specified, NOALTE; for any other combination of LINES and LEN, ALTE; for 328x printers, OPT=NOALTE.
---------	---

Is applicable to terminals that support alternate screen sizes.

Specifies whether the alternate or standard erase write is used. Use of the alternate erase write generally results in a larger device buffer size.

OPT=ALTE indicates that the alternate erase write is used.

OPT=NOALTE indicates that the standard erase write is used.

EBCDIC|BCDIC|EBCDIC,ADD|BCDIC,ADD

Optional.

Note:

This option applies only to IBM 2740 model 1 terminals, and will be ignored if another terminal type is specified.

Default	OPT=EBCDIC
---------	------------

Specifies the type of keyboard being used.

OPT=EBCDIC indicates an EBCDIC keyboard.

OPT=BCDIC indicates a BCDIC keyboard.

OPT=EBCDIC,ADD indicates an EBCDIC keyboard with an adding machine.

OPT=BCDIC,ADD indicates a BCDIC keyboard with an adding machine.

CR|NOCR

Optional.

Default	OPT=NOCR
---------	----------

Specifies whether or not the carriage return is supported by the device being defined.

OPT=CR indicates that the carriage return is a supported feature.

OPT=NOCR indicates that the carriage return is not a supported feature.

EM3270|NOEM3270

Optional.

(Applies to TTY TIDs only.)

Default	OPT=NOEM3270
---------	--------------

Specifies that the TID should use 3270 emulation for a 3101 device. Refer to 3101 Terminal Support for further details.

SHARE|NOSHARE

Optional.

(Applies to VTAM printers only.)

Default	OPT=NOSHARE
---------	-------------

Specifies whether or not a printer can be shared among local copy functions, Com-plete, and other VTAM applications.

OPT=SHARE indicates a VTAM printer-shared mode. When a printer is identified as shared, Com-plete issues a CLSDST for the printer when there is no more output from the application program. The printer is then free to handle any local copy requests that may have been queued while the device was bound to Com-plete. When output is again available for the printer, Com-plete will reacquire the device via a SIMLOGON, print the data, and again issue a CLSDST for the device.

Note:

OPT=ACQUIRE must also be specified if OPT=SHARE is used.

OPT=NOSHARE indicates that a VTAM printer is not available for local copy requests.

ACQUIRE|NOACQUIRE

Optional.

(Applies to VTAM terminals only.)

Default	OPT=ACQUIRE
---------	-------------

Specifies whether or not the VTAM terminal is to be acquired when Com-plete VTAM startup options are initialized or when VTAM support is started with the VTAM START OC command.

OPT=ACQUIRE indicates that the terminal is acquired when Com-plete VTAM support is started (using the VTAM macro SIMLOGON). For additional information on required sysparm operands, see the VTAM startup options in the chapter entitled *Initialization - Com-plete Startup Procedure*.

OPT=NOACQUIRE indicates that the terminal is not acquired when Com-plete VTAM support is started.

COMP|NOCOMP

Optional.

Default	OPT=COMP
---------	----------

Specifies whether or not data streams being sent to 3270-type terminals (incl. printers) are to be compressed by replacing repeated characters with a repeat-to-address order, thereby reducing the amount of data transferred to the terminal.

Most 3270 and compatible terminals support this compression.

OPT=COMP indicates that compression is to be performed.

OPT=NOCOMP indicates that compression is not to be performed.

SCS|NOSCS

Optional.

Default	OPT=NOSCS
---------	-----------

Specifies whether or not the printer is an SCS device. OPT=SCS specifies an SCS printer. OPT=NOSCS specifies a normal printer.

Note that this option is only valid for ACCESS printers. Printers acquired via Com-plete VTAM will have this option overwritten by the information obtained from the bind during acquisition of the device.

IBM|FACOM|HITACHI

Optional

Default	OPT=IBM
---------	---------

In general, all terminals use the same 3270 command codes. However, some 3270 terminals use a proprietary set of codes. If a device you are defining uses Fujitsu terminal command codes, specify FACOM here, while for terminals using Hitachi command codes specify HITACHI.

PRI={0|1|2|3}

Optional.

Default	PRI=1
---------	-------

Specifies the dispatching priority to be assigned to the terminal being defined. The priority specified can be 0, 1, 2, or 3, where 0 is the lowest priority.

Attached terminals are automatically assigned a priority of 0, forcing them to have lowest priority.

RMC=(c1,c2,...,cn)

Default	The RMC value, if any, assigned in the TIBSTART macro; otherwise, RMC=(1,2,4).
---------	--

Specifies the authorization receiving message class codes to be assigned to this terminal.

Note that if a terminal user logs on to Com-plete, the RMC value assigned to the user ID will override the value specified in the TIB macro.

SCHC=tid

Optional.

(Valid for 3270-type terminals only.)

Default	None
---------	------

Specifies the TID number of the terminal to be designated as the hardcopy device for the terminal being defined.

This keyword operand provides support for the screen-to-hardcopy function of Com-plete, which is available for 3270-type terminals. Details on the use of this function can be found in the Com-plete Utilities documentation.

SMC=(c1,c2,...,cn)

Optional.

Default	The SMC value, if any, assigned in the TIBSTART macro; otherwise, SMC=(1).
---------	--

Specifies the authorization sending message class codes to be assigned to the terminal being defined.

Note that if a terminal user logs on to Com-plete, the SMC value assigned to the user ID will override the value specified in the TIB macro.

STALL={YES|NO}

Optional.

Default	STALL=NO
---------	----------

Specifies whether or not the TIB is stalled when Com-plete is started. No input is accepted from a stalled terminal, but Com-plete does poll the terminal and handles attention interrupts.

STALL=YES indicates that the terminal is stalled when Com-plete is initialized.

STALL=NO indicates that the terminal is unstalled when Com-plete is initialized.

The terminal can be stalled using the STALL operator command or unstalled using the UNSTALL operator command.

TRTABS=trantab

Optional.

Default	None.
---------	-------

Specifies the name of a load module containing the TIB depending translate tables. See copybook CCTR3270 for an example. This copybook maps the system-wide translate tables, which can be modified for general translation, which means CCTR3270 is the default translate table for all TIDs that have no TRTABS specified.

VAL={YES|NO}

Optional.

Default	The VAL coded on the TIBSTART macro, if specified; otherwise, VAL=YES.
---------	--

Specifies whether or not the TIB macro parameters are validated.

Note that assembly of the TIBTAB is faster if validation of the parameters is bypassed. Care must be taken, however, to ensure that the parameters are correct.

VAL=YES indicates that the parameters are validated.

VAL=NO indicates that the parameters are not validated.

CMDEVS Macro

The CMDEVS macro is an optional macro used internally by Com-plete and by the user to force an entry to be created in the device type table for a certain type of device. When a device type code is used in a TIB macro, a device type table entry is created automatically by the TIBEND macro.

In a VTAM environment, a TIB can be created for a VTAM LU that logs on to Com-plete. For the device to be operational, a device table entry must exist for that device. If no TIBs are defined with that device type, no entry exists in the device type table. The CMDEVS macro allows the user to force the entry to be created.

Note:

The CMDEVS macro must be defined before the TIBEND macro.

The format for using the CMDEVS macro is:

```
[name] CMDEVS GEN,TYPE=xxx
```

The arguments are:

name

Optional.

Specifies the CMDEVS statement name; for assembly listing purposes only.

Note that name can be any character string that is a valid Assembler tag symbol.

GEN

Required.

Indicates that the device type specified by the TYPE argument is to be added to the generation/definition of the TIBTAB.

TYPE=xxx

Required.

Specifies the type of device to be included in the TIBEND generation.

Here, xxx must specify a device type code as listed in Terminal Device Type Codes.

Note:

The TIBSTART macro automatically defines the following device types:

3270L, 3278L, 3284L, 3286L, 3287L, 3288L, TTY, BATCH, LU62S, LU62C, FSP.

This means that the CMDEVS macro need not be specified.

TIBEND Macro

The TIBEND macro defines the end of the TIBTAB.

Note:

TIBEND must immediately follow all TIB macro statements and (if any exist) all CMDEVS macro statements.

The TIBEND macro has several functions. Its primary function is to generate a device table entry for each device type that has been defined in a TIB macro. (Additional device types can be defined using the CMDEVS macro.)

There are no operands specified for the TIBEND macro.

The format for using the TIBEND macro is:

```
TIBEND
```

END Statement

The END statement is an Assembler instruction that identifies the end of the source module definition.

Note:

The END statement must immediately follow the TIBEND macro statement.

The format for using the END statement is:

```
END
```

How to Create TIBTAB

After coding the desired TIBTAB, the table must be assembled and link edited into a Com-plete STEPLIB. It is recommended that the TIBTAB be page-aligned for the purpose of clarity during problem determination. To achieve this, use the PAGE linkage editor option.

The new TIBTAB will be available for use the next time Com-plete is initialized.

Note:

The TIBTAB being created must be expressly requested at initialization time by use of the sysparm TIBTAB.

Dynamic Completion of TIBTAB During Com-plete Initialization

VTAM and ACCESS terminals need not necessarily be defined in the TIBTAB load module. Instead, you can maintain a table of TIB definitions which is used by Com-plete during initialization to dynamically create or complete the TIBTAB. In this table, you can define TIBs with unique names, with or without fixed TIB numbers (TIDs). You can specify all parameters described for the TIB macro in this chapter. For details about the appropriate maintenance utility, refer to the description of UUTIL, function TT in the Com-plete Utilities documentation.

During initialization, depending on sysparm TIBTAB, either the TIBTAB load module is loaded or an empty TIBTAB is created. Afterwards, the TIBTAB is completed dynamically using this TIB definition table. At the first step, all TIBs defined with fixed TIDs are built, if the appropriate TIDs are still free. Override takes place only if TIB name and TID match, otherwise the dynamic TIB definition is skipped. At the second step, TIBs defined without a fixed TID are allocated using free TIBTAB entries. If a TIB with the same name already exists, it is overridden by the dynamic definition.

Printout Spooling TIBTAB Considerations

This section discusses printout spooling TIBTAB considerations.

Defining Virtual Printers

Use of the printout spooling facilities may require that TIB entries be inserted in the Com-plete TIBTAB for virtual printers. The following example shows the recommended structure of such entries:

```
TIB 131,VTAM,3288L,NAME=VRTPRI1,DEL=YES,GROUP=(ALL=NO)
```

where:

VTAM is used to define a dummy TIB.

VRTPRI1 is a sample printer name that can be used to access this virtual printer.

DEL=YES must be coded for virtual printers.

Defining Real Printers

Since all output processed for a given form (that is, virtual printer) is routed to a real printer after the appropriate form has been mounted, care should be taken that no message output interrupts the print process of the queue. It is recommended that a TIBTAB entry for a real printer for which the printout spooling facility is to be used be defined as follows:

```
TIB ...RMC=1,GROUP=(ALL=NO)
```

Note that the TID should not be defined as a "screen-to-hardcopy" TID.

Printouts can be sent to any VTAM printer without being specified in the Com-plete TIBTAB as long as an unique destination ID (no group ID) is provided during PSOPEN or when using print functions from the Com-plete utilities. In this case TIB entries are acquired dynamically and released after successful printing.

Batch Output Facility (SYSOUT)

This section provides an overview of the extended batch output facilities provided by Com-plete via printout spooling.

During initialization, Com-plete dynamically creates a TIB entry named SYSOUT (if it does not already exist). All printout routed to this terminal will be automatically spooled to the JES/POWER output queue. The output will be separated using the DYNALLOC/SEGMENT facility.

For example, a printout spool destination of `SYSOUT=X` causes the printout to be routed to `JES/POWER` in `CLASS=X`.

Defining JES/POWER Nodes

Defining TIB entries with `DEL=YES` and `ALTTIB=nnn` (where *nnn* is the `SYSOUT` terminal ID) results in routing all printout for these destinations to the `JES/POWER` queues as described above. However, the `NODE` name generated for the `DYNALLOC/SEGMENT` is the `TIBNAME` of the original terminal. Using this method, printout can be routed directly to any specific `JES/POWER` node.

Note that setting `applmod 85` will cause dynamically generated printers to be allocated with these attributes.

Dynamic Routing

The user exit `ULMSBTCH` can also influence the `JES/POWER` routing parameters.

For example, `ULMSBTCH` could request that printout be spooled to a `JES/POWER` queue by modifying the parameter `ULSTNODE` to contain `NODE01` and `ULSTUSER` to contain `USER01`. `Com-plete` then issues a `DYNALLOC/SEGMENT` with the modified parameters and the printout is routed to the specified node/user ID.

Note:

The `SEGMENT` statement provides a `POWER` listname as well as a `$$ LSTcard` where the different keyword parameters are supplied as below.

```
* $$ LST DISP=d,CLASS=c,FNO=ffff,DEST=(node,userid)
```

The `SEGMENT` expression in this description refers to this `$$ LST card`. For more information on the `ULMSBTCH` exit, see the chapter `Security and User Exit Facilities`.

Logical Output Drivers

This section explains the logical output driver facility available in `Com-plete`'s printout spooling system.

What is a Logical Output Driver?

A logical output driver is a user-written routine that allows modification, extension and/or translation of output data at the time the printout is routed to the physical printer.

The output driver runs under control of `Com-plete`'s `MSGPO` task and is loaded dynamically from the `COMPLIB` dataset(s). It is deleted as soon as its use counter is zero. All operating system functions (`OPEN`, `CLOSE`, `SVCs`, etc.) are available for use. The output driver routine has to be serially reusable since it can be invoked for multiple printouts in parallel.

Note:

The `Com-plete` functions (`MCALL`) as described in the `Com-plete Application Programming` documentation are **NOT** available to the logical driver facility.

The driver can be used for the following:

- Precede the output data with printer-specific escape sequences;
- Add graphics to the printout;
- Translate pseudo-commands from the application to printer-specific commands.

The source library supplied with the installation tape contains 2 examples of logical output drivers (LDRVSAMP and IPDSDRV).

One of the major benefits in adding printer-specific commands and/or escape sequences using a logical output driver is that when your site changes its hardware, only the output driver routine requires modification: all user applications that create printouts can remain unchanged.

Specifying Logical Output Drivers

The name of the output driver associated with a printout can be supplied via the LDRIV keyword of the PSCB used for PSOPEN processing, and it can be changed or added using the functions of the USPOOL utility. Installations wanting to use a general output driver can use the POLDRV sysparm specification. The driver specified in the POLDRV sysparm will be used for all those printouts which have no driver defined explicitly.

Logical Output Driver Interface

The communication area between the MSGPO task and the output driver is mapped by the CMLDRVW macro from the supplied source library, which is passed via register 1 (R1) on entry. Each output driver must have specified the appropriate definition:

```
name    CMLDRVW TYPE=USER.
```

The interface area consist of a fixed header (DRVBUFF-DRVPARM) followed by a variable area, the size of which is dependent on the number of lines and the linelength specified for this printer:

```
varlength = No.lines * linelength
```

The total length of the passed area is provided in the field DRVPARML. None of the fields in the fixed part of the interface area should be used as working fields for the output driver routine.

This interface area will remain active until the end of the printout, or until the driver returns an end-of-work condition (RC=16), and can therefore be used to keep status information for the individual printout.

The driver routine is invoked for each record in the printout, and the following register settings are passed :

R1 address of the workarea (CMLDRVW TYPE=USER)

R2 COMREG

RD 18 word savearea

RE return address

RF entry point.

The field DRVPARMA contains the address of the current record; the length of this record is in field DRVRECL. These fields are set by Com-plete on entry and must be set by the output driver prior to return to the calling module.

On entry, DRVPARMA points to DRVREC within the communication area. However, the output driver must always use the address passed in DRVRECA and not rely on the contents of DRVREC. The individual output line is passed to the output driver prior to any translation of linefeed and/or formfeed characters in the same format as provided by the application that created the printout. The ASA control characters are converted to the appropriate printer control orders after the output driver is invoked.

Return Codes

Code	Meaning
0	<p>unmodified</p> <p>The line is added to the output buffer unchanged. Com-plete does not check for any values supplied in DRVRECA and DRVRECL.</p>
4	<p>record modified</p> <p>The output driver must provide the address of the modified record in DRVRECA and the length of the new record in DRVRECL. The data is added to the output buffer.</p>
8	<p>insert record</p> <p>The output driver must provide the address of the new record in DRVRECA and its length in DRVRECL. The data is added to the output buffer. The original record from the printout is passed to the output driver again on the next invocation.</p>
12	<p>delete record</p> <p>The current record from the printout is skipped.</p>
16	<p>terminate driver</p> <p>The current record is added unchanged to the output buffer. The work area is freed and the use counter of the driver routine is decreased, and if zero, the driver is deleted from virtual storage. Any remaining records in the printout are passed to the printer as if no logical output driver were specified.</p>
20	<p>send data asis</p> <p>The normal output buffering used for printouts cannot take account of data that has to be sent within a single output request: Com-plete's buffering always adds up output lines until the buffer (the size of which is determined by the line length and number of lines) is full. On the other hand, creating bar codes using Intellegent Printer Data Stream (IPDS) commands requires that all data for a page is passed on a single output request. The data pointed to by DRVRECA with the length contained in DRVRECL will be send to the printer without any further modifications by Com-plete.</p> <p>Same as code 8 (insert record) above, but Com-plete will send the data unmodified, including the first byte (not treating it as an ASA control character).</p> <p>Note that this response has no effect on the positioning within the current printout, that is, the next time the logical output driver is called, the same printout record will be provided. It is the responsibility of the output driver to ignore this record (if required) with a delete record response.</p>

Line Printer Daemon (LPD) Protocol

The LPD Protocol is described under the folliwng headings:

- Introduction

- Concepts
- Printing via the Local Workstation
- Printer Definition Using Environment Variables
- Printer Search Sequence
- LPD Spool vs. Com-plete Spool
- EBCDIC - ASCII Conversion
- Logical Output Drivers
- Printing via Printer "Boxes" Supporting the LPD Protocol
- Changing Printer Definitions Dynamically

Introduction

The LPD protocol is a print server protocol widely used in UNIX networks. In detail, this protocol is described by RFC 1179 which can be found on the Internet, for example at <http://www.faqs.org/rfcs/rfc1179.html>.

LPD server software is available from many different vendors for UNIX and Windows platforms.

Com-plete implements the client part of this protocol, allowing you to route printouts to an LPD print queue instead of the Com-plete spool. From the application programs' point of view, it makes no difference where the printout is routed, the API remains unchanged. This implies that any existing application which create printouts under Com-plete can be used unchanged for printing to LPD print servers. There is one limitation, though: Destination lists with more than one destination are not supported through this interface.

Concepts

An LPD print server usually is a (UNIX or Windows NT) computer with printers connected to it. Each printer is represented on the server by means of one or more print queues. The printer drivers are installed on this server. Different queues for the same printer can be used, for example, to use different fonts or formats like landscape orientation.

From the above follows that it takes two parameters to unequivocally address a printer:

- the IP address of the LPD server, and
- the name of the queue on this server.

In Com-plete, a printer is known by an 8-character name, or a number. Two different ways are available to map these printer names or numbers to LPD printers:

1. You can setup a central table for Com-plete (a list of environment variables) in which you define each of the printer names or numbers to be used in Com-plete, and specify the corresponding LPD server and queue name for it, like `printer=server/queue` (see section Printer Definition Using Environment Variables below).

2. In UUTIL UD (personal defaults), every user can specify an LPD server in the field labeled "Server". If this value is set, then every printout created by this user will be routed to this server, and any printer name this user enters in Com-plete will be interpreted as a print queue name on this server.

Printing via the Local Workstation

When a user connects to Com-plete using Com-plete's individual Telnet tn3270 port, then Com-plete knows the current IP address of this user's workstation. If there is an LPD server active on this workstation, then it is possible to route printouts to this LPD server using the current IP address from the Telnet session. This is useful especially in networks with dynamically assigned IP addresses. To make use of this feature, the user must enter an asterisk ("*") in the Server field in UUTIL UD.

Printer Definition Using Environment Variables

The POSIX layer now available in Com-plete supports the concept of environment variables. See *SMARTS Installation and Operations documentation* for details. A convenient way of setting environment variables is to define ENVIRONMENT_VARIABLES=DD:CONFIG in the POSIX parameters (POSIX parameters can be concatenated with the Com-plete SYSPARMS). This tells Com-plete that in your Com-plete JCL procedure there is a statement like this:

```
//CONFIG DD DISP=SHR,DSN=MY.SOURCE(ENVIRON)
```

where MY.SOURCE(ENVIRON) contains the environment variables.

The environment variable setting for an LPD printer must look like this:

```
SAG_APS_LPD_printer=server/queue
```

Where

printer	is the name or number of the printer as used in Com-plete,
server	is the name or IP address of the LPD server, and
queue	is a printer queue on this server. If the queue is omitted, Com-plete uses the default "LPT1".

Example:

```
SAG_APS_LPD_32=my.lpd.server/queue1
SAG_APS_LPD_PRINTER1=my.lpd.server/queuexyz
SAG_APS_LPD_PRINTER2=111.222.333.444
SAG_APS_LPD_PRINTER3=111.222.333.444/lpt3
```

Printer Search Sequence

In general, you can print to both LPD and VTAM printers from the same Com-plete. Com-plete will try to resolve the printer name or number entered in the following order:

1. If the user has an LPD server set in his personal defaults, then every printout created by this user will be routed to this server, and any printer name this user enters in Com-plete will be interpreted as a print queue name on this server.

2. If the printer name or number is mapped by means of an environment variable (see section Printer Definition Using Environment Variables), then the server and queue name from this environment variable will be used.
3. Otherwise, the printout will be written to the Com-plete spool, and Com-plete will then asynchronously try to route the printout in the traditional way, using the definitions in the TIBTAB, VTAM, etc.

LPD Spool vs. Com-plete Spool

Since an LPD print server includes a spool of its own, printouts routed to an LPD print server are not spooled in Com-plete. This also means that if the printout cannot be delivered to the server (for example because the queue name specified does not exist), then it is discarded by Com-plete. If the printout cannot be delivered due to a situation that is likely to exist only temporarily, then Com-plete will retry a reasonable number of times before discarding the printout. When Com-plete terminates, all LPD printouts not delivered are lost.

Copying or moving printouts from the Com-plete spool to an LPD print server or vice versa is not supported in this version of Com-plete.

EBCDIC - ASCII Conversion

All data supplied through the API function PSPUT must be in EBCDIC encoding, and is converted to ASCII by Com-plete. Line breaks are converted into CR LF (carriage return + line feed) as expected by most LPD print servers. An ASA control character '1' ("new page") on the first line of a printout is removed, and every printout is terminated by an ASCII FF (form feed). Applymods 71, 72, 87, and 95 are ignored for printouts to LPD print servers.

SCS printer control data is not supported through this interface in this version of Com-plete.

Logical Output Drivers

Logical output drivers are supported for printouts to LPD print servers, however, in most cases the task that used to be solved by a logical output driver can be solved in a more convenient way by simply using different queues on the LPD server.

If this is not possible, then most likely your logical output drivers need to be adopted, because the requirements to these programs are stronger than they used to be for traditional logical output drivers:

1. A logical output driver is automatically loaded as a Com-plete-resident program when called for the first time for a printout being sent to an LPD print server, therefore, it must be re-entrant. Traditional logical output drivers ran only under control of the Com-plete MSGPO task. When called for a printout being sent to an LPD print server, the logical output driver runs in the same task as the program creating the printout. This means, it can be called by more than one program at the same time.
2. Logical output drivers must be able to run in AMODE 31, and return to the calling program in its proper AMODE.
3. Traditional logical output drivers used to insert escape strings designated for mainframe-oriented print servers. These print servers are bypassed when printing via an LPD print server. Therefore, the logical output driver must now supply the original escape strings as expected and understood by the

printer. In addition, you'll have to setup the LPD queue accordingly ("raw mode") so it will let these strings through to the printer unmodified. Note that the logical output driver still must supply EBCDIC data.

Printing via Printer "Boxes" Supporting the LPD Protocol

Many printer network "boxes" support the LPD protocol directly, so you might be tempted to send printouts from Com-plete directly to the printer. This is possible, but doing so you lose the features provided by an LPD server, mainly the possibility to install a printer driver, and the advantages of a spool. Therefore, Software AG recommends that you use real LPD server.

It is unlikely that printer manufacturers will develop printer drivers for Com-plete, so if you want to send printouts directly to the printer, you'll have to provide control information (so called "escape strings") to the printer in order to select a font, formatting, etc. The printer definition environment variables (see above) can be used to provide an escape string to be sent to the printer before the printout, and another one to be sent after the printout:

```
SAG_APS_LPD_printer=server/queue/escape1/escape2
```

If you use this notation, then "escape1" will be sent before each printout, and "escape2" afterwards.

A number of options exists how to define these escape strings:

1. You can enter escape strings directly into the printer definition. Any period (".") will be translated into the ASCII escape character <ESC>.

Example:

```
SAG_APS_LPD_PRINTER1= my.lpd.server/queuexyz/.&a6L.(s0p12H/.E
```

2. You can define escape strings using separate environment variables starting SAG_APS_ESC_ . These variables can then be referenced in printer definitions.

Example:

```
SAG_APS_ESC_myescape=.&a6L.(s0p12H
SAG_APS_LPD_PRINTER1= my.lpd.server/queuexyz/myescape
SAG_APS_LPD_PRINTER2=111.222.333.444//myescape
```

3. You can define default escape strings to be used with every printer whose definition doesn't specify escape strings explicitly. To do so, assign the default escape strings to the following environment variables:

```
SAG_APS_ESC_DEFAULT_START=
SAG_APS_ESC_DEFAULT_END=
```

4. If a starting escape string is effective for a printer (in any of the 3 ways described above), but no ending escape string, then Com-plete automatically sends the escape string <ESC>E, which is the PCL5 "reset".

Changing Printer Definitions Dynamically

The environment variables described above are read once during Com-plete (or POSIX, to be exact) startup. There are plans for the future to provide a global tool for changing the values of environment variables dynamically, without having to restart POSIX, however, this tool was not available yet when this version of Com-plete was released.

As a temporary solution, a utility program UPRTDEF is available in Com-plete, which allows you to change these values, and also to add new variables. Here's a few notes on how to use it:

- Simply edit the data on the screen and press PF5 to apply your changes.
- Overwriting a variable name adds a new variable without deleting the old one.
- Existing variables cannot be deleted physically. To delete a variable logically, clear its value.

Note that this utility operates only on the active settings in the Com-plete address space, it does not update the file where your original definitions are stored. This means, all dynamic changes you make will be lost after Com-plete is restarted, unless you apply the same changes to the file documentation only.

Multiple Copies of Com-plete

This chapter describes the use of multiple copies of Com-plete at an installation.

This chapter covers the following topics:

- Multiple Com-pletes in One System
 - Multiple Com-pletes in a Parallel Sysplex
-

Multiple Com-pletes in One System

More than one copy of Com-plete can run on the same system at one time. Running multiple copies of Com-plete simultaneously allows:

- Separation of data and functions by department or agency;
- Isolation within large application systems;
- Isolation of Com-plete batch support;
- Greater data integrity and privacy;
- Easy conversion from one release or version of Com-plete to another.

Each copy of Com-plete must have its own SD dataset and spool dataset, thus enabling each copy of Com-plete to function independently. For example, a test version of Com-plete with a new feature or fix can be started without affecting production work on other Com-plete systems, or an entirely new release or version of Com-plete can run concurrently with an older production release, greatly facilitating the conversion.

Installation Considerations

Because nearly all the modules can be shared among multiple copies of Com-plete, a full installation of Com-plete is not necessary for each copy. Only those files and modules containing information unique to each Com-plete system must be duplicated.

The Com-plete system files such as the SD dataset contain information that is unique to each copy of Com-plete, and, therefore, cannot be shared. One set of the following Com-plete files must be allocated, initialized, and loaded for each Com-plete system.

File	Description
COMPLETE.SPOOL	Disk-resident queue for messages and printout spooling data.
COMPLETE.SD	Online text editing and SD storage data set.

For more information on initializing and loading these files, see the Com-plete Installation documentation and the sections dealing with the TUMSUTIL, and TUSDUTIL utilities in this documentation.

The Com-plete load libraries can be shared among all Com-plete systems. When compressing these data sets, however, it is necessary to stop all Com-plete systems that use the load libraries. All other Com-plete files can be shared by multiple Com-plete systems.

For information on sharing the system data containers COMSYSn, see **The System Data Infrastructure** in this documentation.

Sysparm Considerations

The following sysparms require special attention when running multiple copies of Com-plete:

ACCESS-ID

This parameter must be unique amongst all users of the same Adabas router SVC number within the same machine, and if NET-WORK is active in the machine and ACCESS-LOCAL is not set to YES, then it must be unique within the network itself.

INSTALLATION

This parameter must be unique on the system, so that it can tell users which Com-plete they are using. However, no check is made that this is in fact unique within the machine.

PATCHAR

This parameter must be unique, as it identifies the Com-plete uniquely within the machine. If more than one Com-plete is started with the same patch character, the second is terminated with a message to the effect that the patch character is already in use.

SMFRECORDS

It is no longer necessary to have a separate SMF record number for each different record type, as the records are now identified within the prefix as to what they were written for, for example, LOGON, LOGOFF and which Com-plete they have come from. Refer to **Com-plete Accounting Facility** in this documentation for more information.

VTAMAPPL

This parameter must be unique because VTAM fails any attempt for a second OPEN to the same ACB name.

VTAMSIMLQ and VTAMSIMLREL

To maintain a flexible system, these sysparms must both be set to YES to ensure that any particular Com-plete for which printers are specified releases them on request from another VTAM application (for example, another Com-plete).

Multiple Com-pletes in a Parallel Sysplex

General

Parallel Sysplex technology provides a number of advantages, the most important of which are high system availability and workload balancing. This version of Com-plete can be installed as a cluster of two or more instances running on different systems in a Parallel Sysplex and known to VTAM as a generic resource. Users log on to the generic resource name, and the sessions are distributed by VTAM equally among the Com-pletes sharing this generic resource name.

From the installation point of view, in contrast to a single-system installation, the following requirements must be met for a Parallel Sysplex installation:

VSAM Record Level Sharing (RLS) must be used for the Com-plete system datasets and SD files so they can be shared between multiple Com-plete instances.

A VTAM Generic Resource name must be defined and specified by means of the VTAMGENERIC sysparm for all peer Com-pletes in the cluster. Users log on to this generic name instead of the unique VTAM APPL-ID of a single Com-plete. See VTAM Interface in *Chapter 4* for more details.

Shared Datasets

The following datasets must be shared between all peer instances of Com-plete:

- all load libraries in the STEPLIB, COMPINIT, and COMPLIB chains,
- the SYSMAP map library (if any is used),
- the three System Data Containers (COMSYSn),
- dynamic SD files. See **Com-plete Files and Associated User Files** in this documentation for details and the Installation documentation for conversion issues.

Separate Datasets

Of the following datasets, a unique copy is required for each of the instances of Com-plete:

- the spool dataset (COMSPL),
- the COMSD dataset, but it must be used for online dumps only (see **Com-plete Files and Associated User Files** in this documentation for details about dynamic SD files),
- the CAPTURn datasets (if any are used).

COMSYS Data Containers

When sharing the COMSYS data containers, Com-plete must access them using VSAM RLS (record level sharing). In order for this to work, LOG(NONE) must be specified for each of the COMSYS VSAM clusters. Depending on the rules in effect on your system, other parameters like STORAGECLASS may also require different values.

In your Com-plete JCL procedure, add the parameter RLS=NRI to each of the four COMSYSn DD statements.

Startup JCL Procedure(s)

Though some of the required datasets cannot be shared, a single JCL procedure can still be used for all Com-plete instances in a cluster. These datasets should be named in a way so that, e.g., the &SYSTEM or &SYSCONE system variables can be used to refer to different datasets on different systems in the sysplex.

Example:

If the value of &SYSCONE is 'SA' on system A and 'SB' on SYSTEM B in the sysplex, and your spool datasets are named COM.SA.SPOOL and COM.SB.SPOOL, then the COMSPL DD statement in your Com-plete startup procedure could look like this:

```
COMSPL DD DISP=SHR,DSN=COM.&SYSCONE..SPOOL
```

The advantages of using a single JCL procedure are maximum consistency and minimum maintenance effort.

Sysparm Considerations

Sysparm Members

With a few exceptions (see below), all sysparms will have the same values for all Com-plete instances in the cluster. Therefore, to keep maintenance effort low, it is a good idea to define the common sysparms in a shared sysparm member and only the system-dependent sysparms in separate members for each instance. All these sysparm members can reside in one shared PDS. In the shared JCL procedure, this could look like in the following example:

```
SYSPARM DD DISP=SHR,DSN=COM.CONTROL(PARM_ALL)
          DD DISP=SHR,DSN=COM.CONTROL(PARM_&SYSCONE)
```

Members in COM.CONTROL: PARM_ALL:

```
PARM_SA
PARM_SB
```

Sysparms which are likely to have different values for different Com-pletes in a cluster

INSTALLATION (not necessarily)

PATCHAR (not necessarily)

VTAMAPPL (required by VTAM to be different)

Internals

This part of the Com-plete System Programming documentation provides information concerning the internal organization of Com-plete.

This information is organized under the following headings:

- Com-plete Files and Associated User Files
- The System Data Infrastructure
- The Com-plete Task Structure
- Com-plete Resource Usage and Estimates
- Com-plete Accounting Facility
- Modifications to Com-plete Modules
- Com-plete Capture Processing
- Com-plete Servers

Com-plete Files and Associated User Files

Maintenance of the Com-plete system requires knowledge of the files distributed with Com-plete, the files created during Com-plete installation, and user files associated with Com-plete. This chapter describes the functions of file allocation, file backup/recovery, and file compression relocation.

You should write off entire disk packs at periodic intervals so that specific files can be restored as needed. Restoring files to their original location preserves the validity of the VTOC and catalog entries. Depending on the specific requirements of your installation, additional backup/restore procedures may be necessary.

Standard operating system backup and recovery techniques can be used for VSAM, ISAM, and BDAM files.

This chapter covers the following topics:

- COMSD - Com-plete Sequential/Direct Dataset
- Dynamic SD files
- COMSPL - Com-plete Spool Data Set
- CAPTURn - Com-plete Capture File(s)
- COMSYSn - Com-plete System Data Containers
- LOAD - Distributed Load Module Library
- USER LOAD - User Load Module Library
- MAP Library
- PROFILES - Editor Profiles Library
- SOURCE - Com-plete Distributed Source Library
- UDEBUG Profile Library
- UDEBUG Text Card Library
- Edit Source Libraries
- Application-Specific Data Sets
- COMDMP Dump Data Set (VSE only)

COMSD - Com-plete Sequential/Direct Dataset

Use of COMSD, the Com-plete sequential/direct dataset is required.

DD/DLBL	COMSD
File name	COM.SD

Description

The Com-plete sequential/direct (SD) dataset is internally split into two sections - one for application SD files and paging files, and one for Com-plete online dumps. Separate directories exist for these two sections. The first record of the dataset contains central information about the dataset and its two sections.

As an alternative to allocating SD files within the COMSD dataset, you can choose the option to dynamically allocate each SD file as a separate VSAM relative record dataset. See the section **Dynamic SD Files** below.

Allocation

This dataset must be allocated as a VSAM Relative Record Dataset and initialized using the Com-plete BATCH utility TUSDUTIL before it can be used. When deciding on the size of this dataset, you must consider the maximum number and sizes of application SD and paging files that will be allocated and the space required for online dumps. For detailed information about allocation and initialization parameters, refer to the description of BATCH utility TUSDUTIL in this documentation.

A sample allocation and initialization job is provided on the installation tape. You can use this job as a model (see the *Com-plete Installation and Migration documentation*).

Compression

Free space is reused, therefore no compression is necessary.

Backup/Restore

Backup and restoration of the entire SD dataset using VSAM utilities is possible only when Com-plete is not active. Selective restoration of SD files from this kind of BACKUP is not supported. TUSDUTIL can be used to backup SD files with optional deletion of "old" SD files and for full or selective restoration of SD files.

Backup and restore of SD files using this utility can be performed only while Com-plete is active with SYSPARM ACCESS=YES. TUSDUTIL supports restoration of SD files from V44 and V45 BACKUP datasets written by the former Com-plete BATCH utility TUSRSDCM.

Reallocation

There are no restrictions on reallocation of the SD dataset.

Dynamic SD files

Requires DFSMS 1.3 or higher.

As an alternative to allocating SD files within the COMSD dataset, you can choose the option to dynamically allocate each SD file as a separate VSAM relative record dataset. If you opt to do so, COMSD is still required, but is used for online dumps only. In this case, in order to avoid wasting disk space, SDFILES=0 should be specified when initializing COMSD.

DD name: SYSnnnnn (generated dynamically during allocation)

File name:

prefix.\$.tempname .T tid	for temporary SD files,
prefix.STIMER. &sysname	for Com-plete's UTIMER SD file,
prefix. name .T tid	in all other cases.

Where:

prefix	is the dataset name prefix specified by SYSPARM SDPREFIX,
tempname	is the name of the temporary SD file with the leading '&&' omitted,
tid	is the value of the TID parameter of the SDOPEN function, converted into a fixed-length five character presentation (SHR is represented as 00000),
&sysname	is the value of the system variable &sysname,
name	is the value of the NAME parameter of the SDOPEN function.

Description

SD files are allocated dynamically as separate VSAM datasets when the SYSPARM SDPREFIX is present, otherwise COMSD will be used.

The advantages of dynamic SD files are:

the ability to share SD files between multiple instances of Com-plete, e.g., in a Parallel Sysplex. The COMSD dataset cannot be shared between multiple instances of Com-plete.

The ability to use standard SMS features for backup, migration, restore, etc. of each single SD file.

Special caution is required when running multiple Com-pletes using the same SDPREFIX value, for two reasons:

1. Applications sharing SD files might require changes in the serialization mechanism they use, e.g., the scope of an ENQ might have to be changed from 'step' into 'systems'.
2. Applications (e.g., Com-plete's editor UEDIT) might not expect the same SD file to be opened more than once at a time. Unpredictable results including abnormal program termination and loss of data may occur when the same userID invokes multiple instances of such an application at the same time from different systems. The only secure way to use this configuration is when all Com-pletes sharing an SD file prefix are entered through the same Generic VTAM Resource name, thus ensuring uniqueness of each session.

Allocation

You don't have to allocate dynamic SD files, they are allocated automatically by Com-plete during execution of an SDOPEN function and deleted during execution of an SDDEL function.

The parameters Com-plete uses when allocating an SD file depend on the following:

The values of the RECORDS and RECLEN parameters of the SDOPEN function. These values are used to calculate the amount of space required.

The DFSMS DATACLASS, STORAGECLASS, and MANAGEMENTCLASS specified for SYSPARM SDSMSCLASS. You might wish to create extra classes to be used for SD files. Software AG recommends that you define only the absolute minimum of parameters for DATACLASS and STORAGECLASS, allowing DFSMS to choose optimum values for CONTROLINTERVALSIZE, etc.

To minimize system affinity in a Parallel Sysplex, SD file access should make use of the Record Level Sharing (RLS) option. To enable Com-plete to exploit this feature, LOG=NONE must be specified for the STORAGECLASS and Com-plete SYSPARM SDRLS=YES must be specified. Note that the LOG parameter of the STORAGECLASS is available only starting from DFSMS 1.4. For DFSMS 1.3, the DATACLASS parameter SHAREOPTIONS=(3,3) or (4,3) should be specified instead, and SYSPARM SDRLS should be omitted.

Backup/Restore/Reallocation

Com-plete does not provide any special utilities for nor does it put any restrictions on backup, restore, and reallocation of dynamic SD files.

Security considerations

Dynamic SD files are always created, opened, and deleted with Com-plete's ACEE in effect, never with the ACEE of the user. Therefore, authorization should be defined similar to other Com-plete datasets.

COMSPL - Com-plete Spool Data Set

Use of COMSPL, the Com-plete message switching/printout spooling file, is required.

DD/DLBL	COMSPL
File Name	COM.SPOOL

Description

The Com-plete spool data set contains messages and printout spool files written by Com-plete or by user applications.

Allocation

You must allocate the data set as a VSAM Relative Record Data Set ("RRDS"). Due to VSAM restrictions, the record size must be 7 bytes smaller than the CISIZE of the VSAM dataset. The minimum CISIZE for this data set is 512 (record size 505) and the maximum size is 8192

(record size 8185). When deciding on the size of this data set, you must consider the general size of the printouts that will be written to it: the larger the printouts, the larger the record size should be. It must then be initialized with the INIT function of the TUMSUTIL utility with the number of records that it should contain. See the description of the TUMSUTIL utility in the chapter Batch Utility Programs for more information.

The size of the spool data set is highly dependent on your installation's usage of message switching and printout spooling. The standard installation job allocates a 5-cylinder 3380 data set, but you will probably need to enlarge this. Please be aware that the spool data set must contain AT LEAST as many records as the value specified for the maximum printout sysparm MAXPRINTOUT (see the description of this parameter). A sample allocation and initialization job is provided on the installation tape. You can use this job as a model (see the Com-plete Installation documentation).

Compression

Free space is reused. No utility is therefore necessary for the compression of the data set.

Backup / Restore

The entire spool data set can be backed up and restored using standard IDCAMS utilities. The TUMSUTIL utility enables the installation to backup and restore printouts to the data set on a selective basis.

Reallocation

The Com-plete spool data set can be reallocated and moved at will without affecting other Com-plete data sets.

CAPTURn - Com-plete Capture File(s)

Use of the Com-plete capture file(s) is optional.

DD/DLBL	CAPTURn
File Name	COM.CAPTUR

Description

The Com-plete capture data sets are where Com-plete writes all its capture information. Only one data set is in use at any one time: when this data set is full, Capture will begin using another data set, and the full data set is freed for use.

You can define up to nine Capture data sets to Com-plete: CAPTUR1, CAPTUR2.....CAPTUR9. To enable the Capture facility, the sysparm CAPTURE must also be specified (see the chapter Initialization for more information).

Allocation

Capture data sets are allocated as VSAM Entry Sequenced Data Sets (ESDS) with a variable record size of between 32 and 4096 bytes. Prior to use, the Capture data sets must also be initialized using the TUSACAPT utility. This utility is described in the chapter Batch Utility Programs. A sample job to allocate two capture data sets is provided on the distributed source data set (see the Com-plete Installation documentation).

Compression

No compression of the Capture data sets is necessary, as the data is normally written to the data set and copied off, or overwritten by more recent Capture data.

Backup/Restore

It is not necessary to backup or restore Capture data sets in the normal way, because these data sets are normally copied and reinitialized. You can use standard IDCAMS utilities to copy the data from these data sets and reinitialize them using TUSACAPT.

COMSYSn - Com-plete System Data Containers

Use of COMSYSn, the Com-plete System Data Set, is required.

DD/DLBL	COMSYS1	File name	COM.VSAM.SYSTEM.BASE
	COMSYS3		COM.VSAM.SYSTEM.USERDEF
	COMSYS4		COM.VSAM.SYSTEM.CATALOG

Description

The Com-plete System Data Containers hold most of the information describing the environment of one or more Com-plete's. Examples are the Com-plete system messages, the User ID definitions (logon definition plus user profiles, PF keys etc.).

Allocation

Allocate COMSYSn via the VSAM utility IDCAMS function DEFINE. Some initial data is loaded from the distribution tape via IDCAMS REPRO. These data sets are allocated and initialized during the standard installation process. You must review the size of these files based on the amount of installation data that exists on the files. A sample job is supplied on the distributed source data set to allocate and initialize these data sets. Please refer to the Com-plete Installation documentation for more details.

Compression, Backup/Restore

Standard VSAM methods (using IDCAMS) can be used for these purposes; therefore, no Com-plete utility is provided for these functions.

Reallocation

Can also be done via IDCAMS as required.

LOAD - Distributed Load Module Library

Use of LOAD, Com-plete's distributed load module library, is required.

DD/DLBL	COMPLIB/In LIBDEF PHASE,SEARCH=
File Name	COM.LOAD

Description

The Com-plete distributed load library contains the executable load modules, tables, and maps, including the Com-plete nucleus. This file should be the second file in the VSE LIBDEF SEARCH string. Please refer to **Initialization** in this documentation for the data set order under MVS.

Allocation

The Com-plete load library is a standard MVS load module library or a standard VSE library. Please refer to the Com-plete Installation documentation for the necessary space allocation for this data set.

In MVS, the load library contains all single CSECTS required to relink any Com-plete program, as well as all programs dynamically loaded during initialization.

In VSE, this library contains all modules with MEMBERTYPE PHASE and OBJ required to relink any Com-plete program, as well as all programs loaded dynamically during initialization.

Compression, Backup/Restore

Use standard operating system utilities to backup and restore these libraries.

Reallocation

There are no restrictions on reallocation of the load library.

USER LOAD - User Load Module Library

Use of USER LOAD, Com-plete's distributed user load module library, is optional, depending on whether installation-specific load modules are required.

DD/DLBL	COMPLIB/In LIBDEF PHASE,SEARCH= and LIBDEF PHASE,CATALOG=
File Name	COM.USER.LOAD

Description

The Com-plete user load library is allocated as an empty library during installation. The user load library should contain the executable load modules, user exits, tables, and maps, including the Com-plete control program that has been specifically tailored for your installation. This file should be the first file in the VSE LIBDEF SEARCH string or the MVS COMPLIB concatenation.

In VSE, this is the designated map library for UMAP.

Allocation

The Com-plete user load library is a standard MVS load module library or a standard VSE library. During standard installation, a data set of 10 3380-type cylinders is allocated. This should be reviewed based on installation usage.

Compression, Backup/Restore

Use standard operating system backup/restore utilities for this library.

Reallocation

There are no restrictions on reallocation of the distributed user load module library.

MAP Library

Use of the Com-plete map library is required for the UMAP utility.

DD	SYSMAP: VSE LIBDEF PHASE,CATALOG=
File Name	COM.MAPS

Description

The UMAP online utility saves and retrieves maps from the map library.

The Com-plete map library is allocated as an empty library during installation.

In MVS, the map library must also be in the COMPLIB concatenation so that maps created by UMAP can be used by programs.

Allocation

The standard installation allocates 2 cylinders of 3380-type devices for maps.

MVS	The MAP library is a standard MVS load module library.
VSE	The maps are placed in the user load library.

Compression, Backup/Restore

Use standard operating system backup/restore utilities for this library.

Reallocation

There are no restrictions on reallocation of the map library.

PROFILES - Editor Profiles Library

Use of Com-plete's editor profiles library is optional.

Description

The User Profiles used by the editor are created and maintained in the Profile library.

The Profile library is the first source statement library found in either the terminal operator's library ID table or the Com-plete UEDTB1 table that is defined with "\$\$" as the two-character ID. No library is specifically allocated during installation.

Allocation

The Profile library is a standard MVS library with LRECL=80, or a standard VSE library. Because the library is opened each time a Profile is accessed/updated, secondary allocation is allowed.

Each Profile member is roughly 1 to 35 card images. Be sure to provide sufficient directory entries for the users who are expected to use the editor.

In MVS, allocate with RECFM=FB, DSORG=PO, and LRECL=80.

Compression, Backup/Restore

Use standard operating system backup/restore utilities for this library.

Reallocation

There are no restrictions on reallocation of the editor profiles library.

SOURCE - Com-plete Distributed Source Library

DD/DLBL	Not applicable
File Name	COM.SOURCE

Description

The Com-plete distributed source library is a standard MVS source library, or a standard VSE library. You are recommended to leave the members on this library unchanged for reference purposes. When a member must be changed, it can be copied to the user source. The library contains:

- Macros and/or VSE Edecks necessary to assemble user exits or Assembler applications;
- COBOL COPY code and PL/I %INCLUDE code necessary to compile applications in COBOL and PL/I;
- Macros and/or VSE Edecks required to assemble TIBTAB;
- Sample job control and data required to perform Com-plete system maintenance and establish the desired user-written security routines;
- The CMOSTYPE macro/Edeck required by some Com-plete macros.

Allocation

The source library is a standard MVS library with LRECL=80, or a standard VSE library. Secondary allocation is allowed.

This data set is allocated during standard installation, see the Com-plete Installation documentation for more information.

Compression, Backup/Restore

Use standard operating system backup/restore utilities for this library.

Reallocation

There are no restrictions on reallocation of the distributed source library.

UDEBUG Profile Library

The use of the UDEBUG facility is optional.

DLBL	SAGLIB - SUBLIB=COMDBPRF
DD	COMDBPRF
File Name	Not applicable

Description

This dataset contains UDEBUG profiles which can contain UDEBUG commands to enable users to customize their UDEBUG session and which are read when the PROFILE command is issued and at startup for each individual user.

This DD name must point to a Partitioned Dataset with a fixed block record format. It must have a logical record length of 80 and can have any blocksize which is a multiple of 80. The blocksize of the dataset will have a direct impact on the catalog size for UDEBUG and therefore should be taken into account.

Allocation

The source library is a standard MVS library with LRECL=80, or a standard VSE library. Secondary allocation is allowed.

Compression, Backup/Restore

Use standard operating system backup/restore utilities for this library.

Reallocation

There are no restrictions on reallocation of the distributed source library.

UDEBUG Text Card Library

The use of the UDEBUG facility is optional.

DLBL	SAGLIB - SUBLIB=COMDBTXT
DD	COMDBTXT
File Name	Not applicable

Description

This dataset contains text cards generated by the assembler option TEST which can be subsequently read by the UDEBUD READ command to build symbols.

This DD name must point to a Partitioned Dataset with a fixed block record format. It must have a logical record length of 80 and can have any blocksize which is a multiple of 80. The blocksize of the dataset will have a direct impact on the catalog size for UDEBUD and therefore should be taken into account.

Allocation

The source library is a standard MVS library with LRECL=80, or a standard VSE library. Secondary allocation is allowed.

Compression, Backup/Restore

Use standard operating system backup/restore utilities for this library.

Reallocation

There are no restrictions on reallocation of the distributed source library.

Edit Source Libraries

Edit source libraries are standard operating system libraries, but their use by Com-plete requires special considerations and the creation of proper interfaces. This section describes the maintenance considerations of these libraries from a Com-plete point of view and explains how to create additional libraries and define the required interfaces.

Overview

Since edit source libraries are standard operating system libraries, their creation and maintenance has much in common with other operating system libraries not used with Com-plete. There are, however, additional considerations due to the implementation of an online environment and the requirements of Com-plete.

Com-plete is distributed with two online text editors:

- UED, the text editor designed for use with hard copy terminals;
- UEDIT, the text editor designed for use with 3270-type terminals or compatible devices.

The files to be edited by each of these text editors are not necessarily the same. Considerations for each editor are summarized in the following text. Additional details about each editor can be found in the Com-plete Utilities documentation.

Creation of Edit Source Libraries

The creation and implementation of edit source libraries consists of:

- Allocation of the libraries;
- Adding a UEDTB1 entry or using the UL function of the UUTIL utility.

Allocation

Edit source libraries are standard operating system libraries. In MVS, the allocation of these libraries limits which Com-plete text editor can be used. The possibilities are:

- UEDIT - Partitioned files, sequential files, LIBRARIAN libraries, and PANVALET libraries

RECFM=F or RECFM=FB
 LRECL larger than 80
 LRECL less than 256

- UED - Partitioned or sequential files

RECFM=F or RECFM=FB
 LRECL larger than 2
 LRECL less than 247

In either situation, the BLKSIZE of the files is restricted only by the thread size in which the respective text editor is to execute. If a large BLKSIZE is specified for a given edit file, it may be necessary to modify the catalog entry for the text editor in use to a larger region size. For the UEDIT utility, this means modifying the catalog entry for the modules UEBP and UEPDIN; for UED, the catalog entry for UED itself should also be modified.

In VSE, editor source libraries are allocated and formatted with LIBR.

Note that space allocations must be contiguous.

UEDTB1

Source libraries edited by UED and UEDIT can be referenced by name or by a two-character library identification code. UEDTB1 is a module of non-executable code used to define the two-character identification codes assigned to edit libraries. This module is loaded by both UED and UEDIT, and used only when a two-character library identification code is entered to reference a given library. This module is also loaded and used by the utility program UPDS/USERV. If a two-character code is referenced that is not in this table, an invalid request results. UEDTB1 is system-wide in scope and is shared by all Com-plete users.

A full description of the use, creation, and maintenance of the UEDTB1 module is given in the chapter Security and User Exit Facilities.

DD/DLBL statements are not used to allocate edit libraries except in VSE. All edit libraries are allocated dynamically when they are read and when a SAVE or SUBMIT request is issued. In VSE, the library may be allocated with a DLBL and EXTENT, or Com-plete will optionally construct a DLBL and EXTENT from information provided in UEDTB1 or UUTIL-UL.

Note:

A VSAM VSE library must be defined via UEDTB1 or UUTIL-UL.

UUTIL-UL

Each Com-plete user can define up to 24 personal two-character library identification codes using the online utility UUTIL, function UL. Note that each user's library definitions are searched before the system-wide UEDTB1 definitions.

Note:

A VSAM VSE library must be defined via UEDTB1 or UUTIL.

For more information, refer to **UUTIL - User Environment Definition Utility** in the Com-plete Utilities documentation.

Maintenance of Edit Source Libraries

Since edit source libraries are standard operating system files, their maintenance consists of standard techniques used to maintain source libraries: compression for partitioned files, and reallocation/expansion for sequential files. For example, the MVS utility program IEBCOPY can be used to compress a partitioned file either in place or with an unload followed by a compression in place.

Protection

The utility programs UED and UEDIT perform a DYNALLOC on any source library into which a SAVE/READ function is being performed. This causes an ENQ to be obtained on the queue name SYSDSN. If the operation is a SAVE, then the ENQ is established as with DISP=OLD. For a READ function, the ENQ is established as with DISP=SHR. If another terminal user is actively reading or writing the same source library to which a SAVE or READ operation has been directed, a warning message is given, and the terminal operator has the option of waiting, terminating, or retrying. This same convention exists if a batch job that accesses the library being edited is executing; the terminal operator will be prevented from performing a SAVE operation in a library being accessed by another user.

Note:

In MVS, protection as provided by the TSO/ISPF Editor via the queue name ISPEDIT is not supported.

Application-Specific Data Sets

Application programs running with Com-plete can access VSAM data sets as described in the chapter Software Interfaces of this documentation. In MVS, BDAM and ISAM data sets can be accessed using special Com-plete functions.

Declaration to Com-plete

Application programs refer to DD/DLBL names. All DD/DLBL names referenced by application programs must be declared ("cataloged") to Com-plete using the FM function of online utility UUTIL. This declaration includes the data set name, disposition (MVS only), the name of the VSAM user catalog (VSE only), and other information.

For a detailed description of the FM function of the UUTIL utility, refer to the *Com-plete Utilities documentation*.

Allocation / Deallocation

The data set is allocated to Com-plete dynamically when an OPEN request is issued against the appropriate DD/DLBL name. The data set is deallocated when it is closed explicitly using the CLOSE or BATCH subfunctions of UUTIL-FM.

Compared to permanent allocation of data sets to Com-plete, this mechanism provides maximum flexibility of data set access by BATCH jobs and for data set maintenance (backup, restore, reallocation, rename, etc.), without needing to restart Com-plete.

For compatibility reasons, it is also possible to allocate data sets to Com-plete permanently by specifying DD/DLBL statements in the startup JCL procedure.

Note:

(MVS only): While Com-plete is active, permanently allocated data sets are available for access by BATCH jobs and for maintenance only with the restrictions defined by the JCL DISP parameter and by the SHAREOPTIONS attribute. Setting a permanently allocated dataset to BATCH status closes the dataset and disables online access, but neither deallocates the dataset, nor frees the SYSDSN ENQ held.

COMDMP Dump Data Set (VSE only)

The use of COMDMP, the Com-plete dump data set, is required.

DLBL	COMDMP
File	COM.VSAM.DUMPFIL

Description

The Com-plete dump data set contains a dump of all relevant storage areas in case of a Com-plete abend.

Allocation

The dump data set is VSAM RRDS (relative record dataset) with a required record length of 4080 bytes. The size depends on the size of the Com-plete partition. At least 30 cylinders are required for a 16M partition.

A sample job to allocate a dump data set is provided on the distributed source data set.

Compression

No compression of the dump data set is necessary, as the data is normally written to the data set and copied off, or overwritten by more recent dumps.

Backup/Restore

You can backup and restore the data set using standard IDCAMS functions.

Reallocation

The Com-plete dump data set can be reallocated and moved without any restrictions.

The System Data Infrastructure

This chapter describes how Com-plete system data are organized. It covers the following topics:

- Introduction
 - Sharing Data Among Multiple Com-plete Nuclei
 - The System Data Access Method (SDAM) API
 - The SDAM Control Block
 - SDAM Views
-

Introduction

Com-plete and user profile data are stored on System Data Containers (SDCs) that are VSAM (KSDS) clusters accessed through the System Data Access Method (SDAM) API. Com-plete currently uses the following four SDCs:

Internal Data Set ID	VSAM DDname	Record Size	Nr. of Records	Contents
BASE	COMSYS1	4089	58	Vital Com-plete data
USERDEF	COMSYS3	2041	150	User profiles (User ID, menu,..)
CATALOG	COMSYS4	313	80	Com-plete program / file catalog

Notes:

1. The Nr. of Records is the approximate number of records added to the SDCs during the installation process and may differ from SM to SM.
2. The SDCs, especially USERDEF, can be heavily accessed data sets, depending on the number of users logged on and the type of work they carry out. You are therefore recommended to monitor access against those data sets and place them on a low-contention volume / disk pack to guarantee optimum performance.
3. The space required for the USERDEF SDC is dependent mainly on the number of User IDs defined. As this tends to be the least "static" SDC, its VSAM allocation status should be reviewed with IDCAMS LISTCAT even more often than for the other SDCs. When the report shows many extents, the primary allocation should be increased. With a high number of CI/CA splits, reorganization of the file should be considered as soon as possible.

SDC data records are structured as follows:

Position 1 to 16	The key, consisting of
	-Physical Record ID (binary halfword)
	-System ID (one byte alphanumeric) with a value of x'00' (global record) or the local ID (patch character).
	-Object ID (twelve bytes alphanumeric), e.g. the User ID, message id, ..)
	-Subrecord ID (one byte binary) serves as a sequence number, should there be more than one record for an object ID be required.
Position 17 to 57	Various SDAM control information
Position 58 to nn	where nn represents the maximum record size depending on type of data and the Data Container it is kept on. Only this portion of the record is transferred from/to the requestor of a SDAM function.

Sharing Data Among Multiple Com-plete Nuclei

The SDCs can be shared among two or more Com-pletes. However, the VSAM shareoptions must be changed from the installation default value of (2 3) to (3 3) to allow concurrent updates from all Com-pletes.

The System ID, described in the table above, controls the individual data record's scope. The patch character (set by sysparm PATCHAR) defines the local system ID for every Com-plete nucleus. Switch Applymod 81 on to force Com-plete to look for local data ahead of global records. With Applymod 81 off, local data will be ignored. Note that having Applymod 81 switched on will approximately double access rates to the SDCs, as Com-plete will try to find a local copy of every record read before considering the global one.

This allows users and the system administrator to choose between common environment definitions for all Com-pletes and different profile definitions on one or more systems. Using shared data allows installations with the need to operate more than one copy of Com-plete to greatly reduce administration effort as well as disk space, and reduces data redundancy.

When opting to share the data set(s) please take the following into consideration when choosing the VSAM shareoption:

- Shareoption 3 induces additional VSAM overhead to the system compared to SHR(2), but on the other hand does not fully guarantee data integrity. This should only be used when maintenance is applied to the SDC(s) from a central location (that is, a specific Com-plete).
- Shareoption 4 guarantees full data integrity on the cost of even more VSAM overhead. Nevertheless, this should preferably be used to be 100 % proof. Additionally, note that VSAM will NOT allocate additional extents for a component defined with SHR(4) when one fills. You are strongly encouraged to reserve a number of spare blocks to accommodate unavoidable additions to the SDC(s), again especially the USERDEF dataset.

The System Data Access Method (SDAM) API

The calling sequence is:

```
SDAM (SACB, i/o_area)
```

The SACB (SDAM Control Block) is mapped by CMSACB (BAL), COBSACB (COBOL), and PLISACB (PL/I).

The SDAM Control Block

Field	Usage and possible value(s)
SACRCOD	Return code. See the Messages and Codes documentation for SDAMnnnn messages, where nnnn is the SDAM return code.
SACFDBK	Feedback information. When non-zero, describes the error (SACRCOD) further (that is, VSAM returncode/feedback, Adabas return code, ..).
SACMESS	Message as seen in the Messages and Codes documentation.
SACWORK	Work area for SOFTWARE AG internal use
SACSYST	The System ID (patch character). Low-value means global, otherwise local to the system specified here.
SACOBID	Object ID (that is, item name). This actually describes the specific item within the group (SACSYST / SACVIEW), for example, the message ID, User ID, ...
SACSUBI	Sub-identification. For items spanning more than one record or groups of records for the same object (SACOBID), this specifies the actual item within the group of items.
SACVIEW	A View name that is required to access data through the SDAM API. More than one logical record can be held within one physical record. This allows related data to be kept independently but tightly coupled. Every view can be manipulated independent of other data held in the same physical record, but nevertheless results in performance gains when, for example, more than one view is accessed in a program due to the fact that the underlying physical record has to be read only once and the problem program receives just the logical part it requested. However, a view name can be assigned to entire records as well. A list of available views can be found at the end of this chapter.

Field	Usage and possible value(s)
SACOPER	Operation Code.
	OPEN Set up the environment, allocate workareas etc.
	CLOSE Clean up the environment, free workareas etc.
	READ Retrieve a record
	WRITE Store a new record
	UPDATE Replace an existing record
	DELETE Remove an item from COMSYS
	POINT Specify starting point for sequential retrieval
ENDREQ Terminate sequential processing	
SACFLG1	Processing Option(s), an array of 8 elements.
	SYSOK Retrieve data for SACSYST only, ignore applymod 81 / 82.
	NEWCOPY When the current call (either logical or physical) results in access to the same physical record as manipulated with the last call, the request will normally be satisfied from internal buffers and will not result in a physical read to the storage media. NEWCOPY will force a physical read.
	FULLKEY For sequential operations with a non-zero SACGENL specifies that positioning is to be performed using the full key and not the partial key as specified by SACGENL.
SACTSTM	When reading a record this will contain the record's time stamp. This field is input only.
SACARLN	Length of the area specified as the second input parameter to SDAM, the record area.
SACOCCR	For tabular items specifies the index of the specific item to be accessed (for example, a message record contains the text for various languages, the language is the index into this array).
SACGENL	For sequential retrieval specifies the generic key length. When the (part of) the key specified via SACVIEW, SACOBID, etc, does not / no longer matches the key of a record retrieved by the access method, logical EOF will be signalled.

SDAM Views

The Com-plete distribution source library contains macros describing the record layout for the various records accessible through the SDAM API. The macro names are SDAVxxxx, where xxxx represents the hexadecimal physical record ID behind the VIEW definitions. The following table gives an overview of the views currently available:

View Name	Physical ID	Purpose
USERID	x0080	Com-plete user definition record
USERPROF	x0080	Com-plete user profile record
USERLAST	x0080	Com-plete user's last defaults for UQ, UEDIT, ...
CUSTDATA	xF000	For your use to store site-specific data

The Com-plete Task Structure

When a program is to be initialized, a Com-plete Unit of Work Control Block (CUOW) is allocated and checks are made to insure that the user is allowed to run the particular program. Once this has been done, the work is dispatched as described in this chapter.

This chapter covers the following topics:

- Dispatching (Task Selection)
 - Thread Selection and Reservation
 - Relocation
 - The Quiesced State
-

Dispatching (Task Selection)

Task selection must first determine which task group to use for the work. If no task group is explicitly assigned to the program in its catalog entry, the DEFAULT task group will be used. Dispatching is achieved by ENQuing and DEQuing the Terminal Information Block (TIB) from queue to queue. The Processor Group Control Block (PGCB) has four queues associated with it, one for each possible TIB priority. The TIB is ENQued to the appropriate PGCB queue. If the system is active, it is likely that the TIB will be selected from a queue by one of the active tasks. If this is not the case, the first available task is found by searching the chain of Processor Control Blocks (PRCB) and posting its appropriate task active. The PRCB running the unit of work reserves the required thread and runs the work.

There are cases where the use of certain system functions can cause a program to have affinities to certain operating system tasks. When this occurs, the work is ENQued directly to a PRCB related queue for the task in question. If and when the condition for the affinity no longer exists, the TIB can once again be ENQued to the PGCB queue where it can be processed by any of the tasks in the task group. Work with an affinity is totally dependent on one task and must wait for that task to complete any other work it might be doing. For this reason, affinities should be avoided at all costs.

For OS type systems currently, the only reason a task will have an affinity to a task is when that task issues an OPEN. This is due to the fact that the equivalent CLOSE must be issued on the same operating system task. It is also possible to indicate in a program's catalog entry that it must run with affinity where it would not be possible for Com-plete to internally detect that this is necessary.

Thread Selection and Reservation

Thread selection first involves finding the thread group within which the program will run. If no thread group is explicitly assigned in the program's catalog entry, the DEFAULT thread group will be used. Once the thread group has been found, an eligible thread sub-group must be selected for the work. The first sub-group found that provides the required amount of storage below the line will be the sub-group where the work will run. The sub-groups are searched from the one with the smallest amount available below the line to the one with the largest amount available. Once the sub-group is selected, an attempt will be made to reserve a free thread within that sub-group. If this is successful, the program starts executing. Otherwise, the program is queued to the thread sub-group and will start executing as soon as a thread of

this sub-group becomes available.

While competition for threads should be avoided where possible by allocating sufficient threads in the system, it can still occur. When a non-relocatable program is about to continue its execution after it was rolled out of its thread, it must be rolled back into the same thread. For this reason, once a unit of work has been selected by a task, it must reserve the thread where the work will be carried out. If the thread is busy, the work will be queued to the thread and will only be carried out when the previous work in the thread has finished.

In this case, the task which originally dequeued the TIB is free to go and do other work once the CUOW has been queued to the thread. If the CUOW in question has an affinity to the task, it must not only wait for the thread to become free, but also for its associated task to become free again at the same time. In a busy system, programs running with affinity could suffer relatively long delays waiting for both its thread and task to be available.

Relocation

Relocation is only important if there are an insufficient number of threads to service the maximum number of users who will be concurrently active. The logic here has not changed in that the thread will be prepared for relocation on rollout and the connection with the thread broken. When they are to be rolled in again, the thread selection takes place in the same way as described earlier.

The Quiesced State

The primary purpose of the QUIESCE state is to enable users currently working to finish what they are doing in an orderly fashion but preventing new work from being started. When the system has been quiesced using the QUIESCE operator command, the following will occur:

VTAM will accept no more requests to start sessions with the Com-plete in question. VTAM sessions started before the QUIESCE command was issued will be unaffected.

Attempts to start new sessions via access, be they from batch or some other source will be rejected with an Adabas response code 148 to indicate that the requested node is not active. Access sessions started before the QUIESCE command was issued will be unaffected.

Users already logged on can continue working. However, they will receive a warning message each time they return to their COM-PASS-pass menu indicating that the system is quiescing and that they should finish their work and log off

Once a QUIESCE has occurred, Com-plete must be terminated before normal service can be resumed.

Com-plete Resource Usage and Estimates

This chapter gives an overview of how Com-plete uses the various system resources and shows how you can estimate the amount used, based on Com-plete's needs and other factors at your installation. Depending on the type of resource, a shortage of that resource will under normal circumstances be handled by Com-plete. In some cases, resource shortage does not affect the operation of Com-plete; however, in others, it can mean that certain functions cannot be performed until sufficient storage becomes available. This is discussed in more detail at the start of each section.

Note:

The storage estimates given in this chapter refer to the base level of Com-plete 5.1. Future maintenance may cause these estimates to change. Please refer to the relevant documentation updates issued with each maintenance level.

This chapter covers the following topics:

- Virtual and Real Storage
 - The Roll Subsystem
 - The Com-plete Spool Data Set
 - The Com-plete Sequential/Direct Data Set
 - The UDEBUG Buffer Pool
-

Virtual and Real Storage

A number of mechanisms exist in Com-plete for managing storage. Each is described here, together with the various uses they are put to. This will give you an idea of how to utilize your available storage better and help you understand some of the performance issues and integrity problems associated with the storage management under Com-plete.

Real Storage

It is possible for Com-plete to be swappable. This is achieved by setting `applymod 86` at the startup of Com-plete. This can be useful to avoid test systems from having a detrimental effect on the system. Please note that as soon as ACCESS is started, Com-plete is made non-swappable and will remain non-swappable until the job is terminated. This is because Com-plete becomes a Cross Memory Server and to do this, it must be non-swappable. Stopping and starting ACCESS will not affect this, because the Cross Memory environment is never actually shut down when this is done.

Com-plete Savepool Areas

The lowest level of storage used under Com-plete is the Savepool element. These are elements used as saveareas for nucleus modules. For this reason, these must be allocated correctly from the start, because if they run out, depending on which part is affected by the failure, Com-plete is likely to go down immediately or at least in the following time. This is because it is not possible to expand this Com-plete resource as they are at such a basic level.

Savepool areas can be acquired by Com-plete routines with very little overhead. They are of a fixed length and use the same techniques as the Fixed Buffer Pool Manager to get and free the buffers. They are accountable in that usually, if an abend occurs to a task which has acquired a savepool area, the area can be returned to the available pool and will not be lost.

They can be allocated above and below the line based on the SAVEPOOL and SAVEPOOL-ANY sysparms. When it is possible, a savepool area from the pool which exists above the line will be used. When no such savepool is allocated or the allocated pool is empty, the system will allocate a savepool entry from the pool allocated below the line. When this is exhausted, unpredictable errors may occur depending on what point in Com-plete's processing the failure occurs. For this reason, great care must be taken in allocating the savepool areas.

Due to the transient nature of SAVEPOOL usage, it is impossible to estimate the exact amount of SAVEPOOL entries that will be required for a given installation. The safest recommendation that can be given would be that 5 savepool areas be allocated above and below the line for each active Com-plete task in the system (including system tasks).

It is possible to monitor the usage of the savepool areas using the UTIL MO subfunction SP. These should be monitored over the normal span of activity for the installation. For some, this will be every week, for others it may be every month. Once a full set of figures is available, the worst possible figures can be used as a guideline. Of course, an increase in the workload will again change this. It is therefore always wise to leave a little room for movement and monitor the performance of the savepool areas on a weekly basis.

Com-plete Fixed Buffer Pools

The current version of Com-plete uses a fixed length buffer pool mechanism to manage the most commonly used Com-plete buffers. This opens up new scope for 24-hour operation, as well as improving the availability of Com-plete, as fragmentation cannot exist as it can with variable type buffer pools.

This mechanism enables the allocation of buffer subpools below the 16M line, above the 16M line and in ESA Data Spaces.

When a requested size does not exist in a subpool because the size is greater than the largest subpool element size, the logic causes a new buffer subpool to be allocated with the required size to satisfy the request. This facility can be deactivated on a buffer-by-buffer basis: no new buffer subpool is then created and the request rejected.

For each buffer pool allocated, a chain of buffer subpools will exist for that specific buffer pool. Each subpool represents one specific size / location combination. For example, a subpool can exist with size 1k below the line and above the line, satisfying two logically different types of request. Error handling is as described above with expansion of subpools, subject to storage availability provided for all buffer pools.

The COMSTOR Buffer Pool

When the Com-plete COMSTOR function is used, it generally means that many areas of the same size are allocated in COMSTOR storage depending on the application usage profile. As such, it made sense to use a fixed length buffer pool to handle these requests, as the buffer pool could then be tailored to meet the demands of the various COMSTOR area sizes required.

This provides the following advantages:

- COMSTOR no longer becomes fragmented, which previously called for an over allocation of COMSTOR to ensure that requests were not eventually failed.
- COMSTOR can take advantage of the expansion and contraction capability of the fixed buffer pool manager. This means that even where too little storage has been allocated, the storage can still be made available if there is enough space in the Com-plete region. Both of these factors ensure that COMSTOR need never be a consideration and will never make it necessary to stop and restart Com-plete.
- Storage can be located based on the usage of the applications. Some old applications require that the COMSTOR storage areas allocated for them be below the line due to ECBs being located there. This can be handled by allocating a COMSTOR subpool below the line for these applications. Newer applications can then take advantage of COMSTOR areas above the 16M line and even in ESA data spaces where the COMSTOR area is not being used to contain ECBs.

The COMSTOR buffer pool is built using the COMSTOR-BUFFERPOOL sysparm. The size of the element for each subpool defined determines the largest COMSTOR area size which can be provided from that subpool. Of course, should a smaller size be requested and no smaller subpool is available to handle that size, it too can be satisfied. The number of elements for the subpool determines how many COMSTOR areas are to be initially allocated. Should you wish to limit it to that, you must simply specify "0" for the number of elements by which it should expand if necessary. Otherwise, the subpool will be expanded as necessary to cope with the demands placed upon it. However, no new bufferpool is created if you request a larger COMSTOR area size than the biggest one defined in the sysparms

The location of the subpool storage will depend very much on the usage to which the allocated COMSTOR areas will be put. Where an application program has an ECB within the COMSTOR area and it runs in 24 bit mode, the area must be below the 16M line. When the application runs in 31 bit mode, the COMSTOR area can reside above the 16M line. When the application program does not have an ECB in its COMSTOR area, the subpool from where it will be gotten can be allocated in a data space on ESA capable systems.

Note that for the COMSTOR buffer pool, an element size can only exist for one subpool regardless of the variations in the location of the subpool. This is the case simply because the COMSTOR facility will always allocate the COMSTOR area in the most widely available storage subpool. This means that a subpool built in a data space will be used before one built above the line and so on. Therefore, if a second subpool of the same element size were built in a different location, it would probably never be used.

Apart from the COMSTOR subpools which are built for application program usage, COMSTOR also builds a subpool for the control of the COMSTOR facility in general. This is built based on the total number of elements allocated based on the provided COMSTOR-BUFFERPOOL parameters. Where a subpool with similar attributes to that required by COMSTOR is explicitly allocated by the user (i.e., LOC=ANY, ESIZE=72), this subpools element count will simply be increased by the number of entries which COMSTOR expects to use.

Storage Key of Buffer Pool Subpools

A facility has been introduced to enable the allocation of fixed buffer pool manager subpools in a storage protect key other than that used by Com-plete. Effectively this is another distinguishing factor for a subpool, therefore, a number of subpools can now exist within the same buffer pool with the same element size and location, however, each with a different storage protect key.

The default for the majority of Com-plete's fixed buffer pools is to allocate the subpool storage in Com-plete's key, however, the Adabas Interface requires storage which exists outside of the thread in the thread's key, in order to roll a user program out over an Adabas call. For this reason, the Adabas interface builds a buffer pool with a number of storage subpools in different storage keys. Refer to the discussion about the Adabas interface for more information. It is possible that other Com-plete subsystems may build storage subpools with different storage protect keys in the future.

The Com-plete Unit of Work (CUOW)

The primary descriptor for work in the Com-plete system is the Com-plete Unit of Work control block or CUOW. This control block is built in a buffer in the General buffer pool when a user program is started and exists until the user program terminates. The CUOW contains all information related to the user program.

Thread Groups and Sub-Groups

A user program can be catalogued to run in a specific thread group which must be defined at start up in the sysparms. If a program is not allocated or it has no thread group associated with it, it will run in the DEFAULT thread group *if* a thread sub-group exists within the group large enough to run the program.

For each thread group, one area is required for the Thread Group Control Block (TGCB) while one Thread Subgroup Control Block (TSCB) is built for each sub-group. Each thread within the sub-group is described by a Thread Control Block (THCB) as in previous releases of Com-plete.

It should be noted that the above control blocks (i.e. TGCB, TSCBs, and THCBs) are acquired from the General buffer pool. Previously THCBs were linked with the Com-plete nucleus and followed the system THCBs. The system THCBs remain where they always were but thread THCBs are no longer linked into the nucleus nor will they be allocated beside each other. It should also be noted that these changes include provision for the dynamic reconfiguration of the system which will give sites the ability to add or delete threads and/or thread sub-groups. This means that in the future, THCBs may disappear during the lifetime of a Com-plete run.

Task Groups

In some cases in the documentation, tasks and task groups will be referred to as processors or processor groups. You will also notice that UUTIL MO functions and operator commands related to tasks start with the letter 'P'. The reason that the task related control blocks are often referred to as processors in the documentation is a simple one of naming conventions. When function names and operator commands were being created, the T in 'thread' and the T in 'task' frequently caused the same name to be generated. For this reason, any task related function or control block is prefixed with 'P' which stands for processor. Generally speaking, where the term 'processor' is used, it can be substituted with 'task'.

Much like the thread groups, one or more task groups is allocated at Com-plete start-up. If a program has a task group associated with it in its catalogue entry, the program will run in that task group if it exists. If the program is not catalogued, or has no task group associated with it, it will run in the DEFAULT task group.

For each task group, a Task Group Control Block (TGCB) which for each task within the task group, a Task (Processor) Control Block (PRCB) is allocated. These control blocks are chained from the PRCB. Both PRCBs and the PGCB are allocated from the general buffer pool.

It is possible to add and delete tasks while the system is running through the TASKS operator command. It should be noted that for performance reasons, when a PRCB has been allocated for a task and the task is subsequently deleted, the PRCB is not actually freed. This can only occur when the task group itself is deleted which occurs at EOJ. These so called dormant control blocks can subsequently be reused if more tasks are added to a task group at some future point in the life of Com-plete.

Virtual Storage Usage

The following gives an overview of the major virtual storage areas in the Com-plete address space / region. These are the areas to consider, e.g., when planning for the number of threads to allocate. Experience shows, it is still hard to calculate the exact amount of storage that will be used. E.g., it may be difficult to tell how many I/O buffers are allocated for each of the datasets when they are opened, you don't know in advance which bufferpools will expand, etc. Software AG recommends that you do a rough estimate and start with a configuration that leaves 20-30% of your region below the 16M line free. Then, while Com-plete is running and the maximum number of users is active, use USTOR function ASU to determine the real address space utilization. If you then find that there is a good reserve, you can increase the number of threads in one or more sub-groups.

Thread Storage

Typically, the biggest part of all storage in the Com-plete address space is used by the threads. Each thread in a given sub-group occupies the same amount of storage below the line, as specified by sysparm THREAD-GROUP. Each thread, independently of thread group and sub-group, occupies the same amount of storage above the line if specified by sysparm THSIZEABOVE.

Storage Occupied by Load Modules

- Com-plete nucleus modules;
- Com-plete server modules;
- RESIDENTPAGE modules;
- PGMLOOKASIDE modules.

The location of the modules is defined by their RMODE attribute, except for PGMLOOKASIDE programs, which are always loaded above the line where possible.

Terminal Table (TIBTAB) Storage

The TIBTAB can be assembled and linked, from which the size of the TIBTAB can easily be seen, or it can be dynamically generated for which the amount of storage used must be calculated. The size of each TIB is currently 192 bytes. This version of Com-plete allows the TIBTAB to reside above the 16M line, controlled by means of the load module's RMODE attribute or the sysparm TIBTAB=ANYnnnnn.

Savepool Storage

The size of one savepool element is currently 208 bytes.

Storage for Fixed Buffer Pools

The amount of storage used is the actual buffer storage itself, plus some storage for control blocks. Once successfully initialized, Com-plete wherever possible obtains storage already allocated from the buffer pools. Of course, if a buffer pool must be expanded, this storage is again requested from the operating system.

I/O Buffers and Control Blocks

- for Com-plete's own datasets;
- for VSAM and other datasets used by application programs.

Usually, control blocks and buffers for VSAM files are located above the 16M line, those for all other files below the line.

Natural Buffer Pool Manager

The Natural buffer pool manager simply allocates storage as specified in the sysparms. A message is issued indicating how much storage has been used and where it was obtained. Unless forced below the line via a sysparm, the Natural buffer pool storage is obtained above the line.

Control Blocks of the Operating System and Other Software products

- TCBs, RBs, etc.;
- RACF/ACF2 related control blocks, e.g. one ACEE per user;
- etc.

General Buffer Pool Usage

After initialization, apart from the storage requirements that are specifically allocated at startup, all Com-plete requests for storage are resolved from this buffer pool. This includes:

- Short term working storage requests;
- Medium term requests; this storage is held for the duration of a transaction;
- Long term requests.

There are many different types of working storage areas obtained. Size, number, and location of these areas heavily depend on various factors, making it almost impossible to calculate them exactly. Instead, you are recommended to start with a general buffer pool configuration with a basic number of buffers of 64 bytes, 128 bytes, 256 bytes, 512 bytes, 1 Kbyte, 2 Kbytes, 4 Kbytes, and 6 Kbytes, below and above the line, where applicable, for all of these sizes. For the basic numbers of buffers, reasonable values to start with can be either your values from the previous version of Com-plete (if you are upgrading to a higher version), or you can start with a configuration shown in the following example:

```

BUFFERPOOL=( 64 , 100 , 100 , ANY )      ( * )
BUFFERPOOL=( 64 , 100 , 100 )
BUFFERPOOL=( 128 , 100 , 100 , ANY )
BUFFERPOOL=( 128 , 100 , 100 )
BUFFERPOOL=( 256 , 100 , 100 , ANY )
BUFFERPOOL=( 256 , 100 , 100 )

```

```

BUFFERPOOL=( 512 , 100 , 100 , ANY )
BUFFERPOOL=( 512 , 100 , 100 )
BUFFERPOOL=( 1K , 100 , 100 , ANY )      ( * )
BUFFERPOOL=( 1K , 100 , 100 )           ( * )
BUFFERPOOL=( 2K , 10 , 10 , ANY )
BUFFERPOOL=( 2K , 10 , 10 )
BUFFERPOOL=( 4K , 10 , 10 , ANY )
BUFFERPOOL=( 4K , 10 , 10 )
BUFFERPOOL=( 6K , 10 , 10 , ANY )
BUFFERPOOL=( 6K , 10 , 10 )

```

Since they are related to other sysparms, the allocation parameters for the buffer pools marked with (*) may be changed internally during startup.

If you then monitor the buffer pool statistics, you can easily determine the buffers that should be increased or decreased in size or number.

The Roll Subsystem

While this version of COMPLETE removes the limit of a maximum of 16 threads, the number of threads is still limited by the amount of storage available in the address space. In general, the number of users will exceed the number of threads, so threads have to be shared between users.

When a user program has reached a certain point in its processing, for example a conversational terminal write, it no longer needs to reside in the thread, as the conversational write will take a relatively long period of time to complete. In this case, another user can use the thread. However, the current user's data must be saved somewhere. This section gives a definition of the terms used to describe the Com-plete method of doing this.

Com-plete Rollout/Rollin Processing

Currently Com-plete can save the image in a thread buffer called the "roll buffer". This saving of data is known as a "rollout".

When the user responds to the conversational write, the program copy must be found and moved back to the thread to continue execution. Again the image is copied from the roll buffer back into the thread. This process is referred to as a "rollin".

Com-plete Roll Buffers

In many installations, the sizes of the thread images which are rolled out from thread tend to be very consistent in a production environment. This means that with the provision of a few subpools of the required sizes, a roll buffer pool can be provided which will satisfy the vast majority of rolouts from thread. When a roll buffer pool can be used in this way, there are a number of advantages.

- The allocation and freeing of the buffer is infinitely quicker than for the variable buffer pool which can have a major performance impact where a lot of rolling is taking place.
- No fragmentation takes place, thus providing 24*7 functionality.
- The subpools can reside in a data space, thus freeing up more of the Com-plete region for other storage.

The fixed roll buffer pool is allocated based on the ROLL-BUFFERPOOL sysparm. The element size is the amount of storage available in a given subpool into which the roll subsystem can copy a thread image. This can best be determined by the second roll activities screen (PF11) in UCTRL which provides a map of the sizes of the thread images which are being rolled out. If the load is consistent over a run of Com-plete, this could be used to estimate the most appropriate subpool sizes which should be used for the fixed Roll buffer pool.

The number of elements for a subpool indicate how many thread images the subpool can accept and again, once the number to expand by is not set to 0, it will expand to handle the load. If the subpool is allocated in a data space, this should not pose a major problem. However, if the subpool is allocated in the primary region, due to the element sizes involved, the Com-plete region may fill very quickly.

As the location of these subpools is of no concern to users of the system, they should be allocated in the most available area which is the default. This means that on ESA capable systems, they will be allocated in a data space while on non-ESA capable but XA capable systems, they will be allocated above the 16M line.

Gets and Frees from this buffer pool are extremely efficient using 370 instructions to serialize access to the storage. By definition, it will never be fragmented so that it will still have the same potential to be used after two weeks of operation, as it would have had after being brought up. Lastly, the storage where the image is held can be allocated in a data space on ESA capable machines freeing up a large part of the primary address space for other work.

The Maximum Number of Rolled Out Images.

The maximum number of rolled out images can simply be calculated as follows.

Maximum Logged on Users * (Maximum No. of Stack Levels +1) +

This is a theoretical maximum: some users will not use or will be disallowed from using the maximum number of Stack Levels.

The Com-plete Spool Data Set

The Com-plete spool data set is a VSAM data set and contains the data for all print out spools in the system and all messages that exist on disk.

Data Set Structure

The MAXPO sysparm indicates the maximum number of printouts and/or messages that can exist in the system. Com-plete reserves this number of blocks at the start of the message data set to hold the control information for a printout. The printout data set must contain at least this number of blocks or Com-plete initialization will fail. If more blocks are available than the MAXPO specification, these blocks will be used for printouts whose data cannot fit in the first record along with the control data.

Printout Structure

During initialization, Com-plete interrogates the directory information (contained in the first n blocks of the spool data set) and constructs an in-store Message Core Queue (MCQ) for each printout requeued. This MCQ information is supplemented by the information contained in the control record on disk. The control record can point to more than one data block, Com-plete also maintains a list of index blocks which are used for free space management within the spool data set, and possible positioning of the

printout via USPOOL commands.

Receiver list

When a printout or message is sent, it is sent to a receiver or a list of receivers. The processing is different depending on whether a list of receivers or a single receiver is specified. When a single receiver is specified, the receiver is remembered by name and therefore the printout or message can be restarted in an environment with a dynamic TIBTAB defined. In the same environment, using lists may result in restart problems, as the list will be remembered using TIB numbers causing unpredictable results with the restart. When the TIBTAB is defined, the restart can normally proceed without problems, as unless the TIBTAB has changed, the various TIB numbers will still relate to the original terminal.

Spool Data Set "Data"

The data written to the spool data set is compressed using repeat to address (RTA) type commands, thus saving space. Lines are written into the data areas until the data area is exhausted, then a new data block is allocated and the next line is started in there. To avoid large searches when positioning printouts, index blocks are built. There will be one index block describing multiple data blocks. The index maintains the page and line number that each data block begins with, along with the data block number. Therefore for large printouts, no time is wasted reading from the start of the data blocks to the end of the data in searching for a specific line number.

Spool Data Set Space Calculations

You must first estimate the maximum number of printouts and messages that you expect to see in a system. Once this is done, estimate the average size of the printouts. For small printouts, the space available in the control record sometimes is sufficient so that no extra data blocks are available. However, for installations who have large "receive" lists and/or large printouts, additional index and data blocks must be allowed for over and above the blocks allocated for the message data MAXPO index blocks.

The Com-plete Sequential/Direct Data Set

The Com-plete sequential/direct (SD) dataset contains application SD files, terminal paging files, and Com-plete online dumps.

Data Set Structure

The space of this data set is split into two sections when it is initialized - one for application SD and paging files and the other one for Com-plete online dumps. The first record of the data set contains central information about the data set and its two sections.

Structure Of SD File Space

The SD file directory starts from the second record. The number of records used for this directory depends on the maximum number of SD files as indicated at initialization time and can be changed only by reinitialization of the data set. Each directory entry is 64 bytes long. Each directory record contains an integer number of entries. When an SD file is added, the entire directory is expanded and an entry is inserted at the appropriate place in ascending order by SD file name and TID number. For paging files, no directory entries exist; all information about

paging files is held in-storage only.

The SD file directory records are followed by the space where SD and paging files are built. These records are used as a pool of unique-length blocks. 12 bytes of every block are required for internal use, so the resulting blocksize is `VSAM RECORDSIZE - 12`.

Every SD file consists of index blocks and data blocks. Blocks for both index and data are allocated dynamically from the pool when new records are written. When an SD file is deleted, all blocks assigned to it are freed and returned to the pool. A paging file is automatically deleted when the creating application program terminates; its blocks are returned to the pool.

The index of an SD file is partitioned into blocks on fullword boundaries. The structure of data blocks can be described as follows:

For Com-plete internal use, 1 byte is added to each SD file record. Depending on their length, records are blocked or split to the blocksize mentioned above.

In case of blocking, an integer number of records is written to each block. The last bytes of each block remain unused, if the blocksize is not an integer multiple of `recordsize+1`.

If `recordsize+1 > blocksize`, records are split. In this case, each record starts on a block boundary. Thus, the last block of each record may remain partially unused.

Dump Space Size And Structure

The second part of the SD dataset is used for Com-plete online dumps. The first record(s) of it contain(s) a dump directory of 32 entries. When 32 dumps exist and another one has to be written, the oldest existing dump will be deleted and its directory entry and space will be reused.

The dump directory is followed by the dump space. Its size is defined when the SD dataset is initialized using `TUSDUTIL`. When deciding on the size of this space, you must consider the size of the largest thread as specified in the `THREADS=` startup option. It is recommended to assign space for 32 dumps of this size. If insufficient space is available to add another online dump, Com-plete will delete the oldest existing dump(s) and reuse this space to write the new one. This results in fewer than 32 dumps being held online.

The UDEBUG Buffer Pool

This buffer pool contains two subpools which in turn can contain three UDEBUG control block types.

- The first is the Com-plete Control Block (DBCCB) which is a control block required by a UDEBUG session which must always be available, that is, not rolled out with the UDEBUG session. It contains elements with a length of 56 bytes from which the Com-plete Control Block areas and the Global Symbol entries are allocated. The allocation and usage of this pool can be managed using the `FB` function of `UCTRL` and the virtual storage required for can be determined using estimates for fixed length buffer pools described in the section **Storage for Fixed Buffer Pools**.
- The second control block is for UDEBUG breakpoints (DBBP). It contains elements with a length of 184 bytes from which the UDEBUG breakpoint areas are allocated. The allocation and usage of this pool can be managed using the `FB` function of `UCTRL` and the virtual storage required for can be determined using estimates for fixed length buffer pools described in the section **Storage for Fixed Buffer Pools**.

- The third is for UDEBUB global symbol entries which must be available to all users.

When a user starts a UDEBUB session, a DBCCB is allocated and when the session terminates, either normally or abnormally, the DBCCB is freed. This means that the number of DBCCBs allocated will be in direct proportion to the number of UDEBUB sessions.

Breakpoint buffers are gotten but never actually freed back to the buffer pool, they are logically freed by UDEBUB to make them available for other breakpoint requests. This was necessary to reduce the cost of serialization. It also means that once the breakpoint subpool has expanded, it will never be contracted. The following discussion therefore relates to the logical getting and freeing of breakpoint buffers.

Breakpoint buffers are obtained implicitly and explicitly. An explicit "get" occurs when the user sets a breakpoint successfully. The freeing of this buffer is a little more complex. Generally speaking, the breakpoint is deleted explicitly by the user or when the UDEBUB session terminates. However, this does not mean that the buffer can be freed. The buffer can only be freed when it can be determined that firstly, the breakpoint is inactive, and secondly, that the breakpoint is not activated while it is being deleted. This generally applies to a situation where the TIB being debugged is different to the TIB on which UDEBUB is running. To avoid costly serialization, these DBBP buffers can only be freed by the debugged terminal.

Even then, it is unlikely that the buffer can be freed. For example, if the breakpoint is for a program which resides in the thread, this cannot be deleted until this user is active in thread again. In the case, the buffer can therefore only be freed once this user has become active and the breakpoint actually reset.

The Roll Buffer

To enable the UDEBUB session to access storage from the user program which is being debugged, the debugged user will be forced to roll out over a BP, as opposed to the normal situation, in which the user is only rolled out when someone else needs the thread. Also, the user must be rolled out to the roll buffer, otherwise the storage will not be accessible by the UDEBUB session. Lastly, no compression of the free queue elements in a thread will be done to ensure that the thread layout is identifiable to the user. This means that the full catalog size of the program being debugged will always be rolled out.

Once the user is successfully rolled out to buffer, the system will ensure that this image is never staged out to ensure that it remains available to the debugging user. This means that the rollbuffer size must be chosen carefully to ensure firstly, that the thread of the program being debugged can always be rolled into the buffer, and that the impact of doing this does not impact the general performance of the Com-plete system.

CPU Usage

As UDEBUB is a debugging tool, the main purpose is to provide functionality with CPU usage a secondary consideration, as this functionality will only be used when testing programs. For this reason, the usage of UDEBUB in a system, particularly through the setting of breakpoints, can significantly affect the CPU usage within Com-plete, even for users who are not debugging or being debugged. Therefore the use of UDEBUB in a production environment must be strictly controlled.

This chapter gives an overview of how Com-plete uses the various system resources and shows how you can estimate the amount used, based on Com-plete's needs and other factors at your installation. Depending on the type of resource, a shortage of that resource will under normal circumstances be handled by Com-plete. In some cases, resource shortage does not affect the operation of Com-plete; however, in others, it can mean that certain functions cannot be performed until sufficient storage becomes available. This is discussed in more detail at the start of each section.

Note:

The storage estimates given in this section refer to the base level of Com-plete 6.1. Future maintenance may cause these estimates to change. Please refer to the relevant documentation updates issued with each maintenance level.

Com-plete Accounting Facility

This chapter describes the Com-plete accounting facility.

This chapter covers the following topics:

- Overview
 - User ID Accounting Block
 - SMF Records
 - SMF Record Contents
-

Overview

The Com-plete accounting facility provides data that can be used to distribute the costs of the teleprocessing system back to the user (terminal user, department, etc.).

Com-plete accumulates user resource utilization information on a user-by- user basis and writes this information to the System Management Facilities (SMF) file. All records are optional. You must specify which records to create and/or not to create by using the SMFRECORDS sysparm.

User ID Accounting Block

The user ID accounting control block is created immediately after a user ID successfully logs onto Com-plete. The information from the logon information block is used to initially create this block.

The user ID account control block is maintained for the duration of the user ID's logon cycle (until *ULOG OFF). Information from this control block is used to create the SMF accounting control records.

Please refer to the delivered Com-plete source copy book CCUAB for details of the contents of this control block and the offsets of the various fields.

SMF Records

The SMF accounting control records contain information about the resources used by a user ID during the life of the user ID cycle from *ULOG ON to *ULOG OFF. The information in each control block is initially taken from the user ID accounting control block.

Five types of resource utilization records can be written to the SMF file:

- The ULOG ON Record
- The Program Termination Record
- The Checkpoint Record

- The ULOG OFF Record
- The User Record

For assembler programs, the accounting records are mapped by member CCCSMF in the distribution source library.

The ULOG ON Record

This record is written after each successful ULOG ON operation. It contains the standard SMF prefix and the following information:

- Terminal Identification number;
- User ID;
- Account number;
- Control status;
- Authorization code;
- Message and printout spooling class codes.

The Program Termination Record

This record is written each time an application program or Com-plete utility program terminates. It contains the standard SMF prefix and the following information:

- Terminal Identification number;
- Thread number;
- User ID;
- Account number;
- Control status;
- Privileged program indicator;
- Abnormal termination indicator;
- Time program was invoked;
- Program name;
- Total CPU time since ULOG ON;
- Total thread occupancy time;
- Total number of EXCPs since ULOG ON. For MVS systems, the SMF record contains the number of SIOs instead of the number of EXCPs. The number of SIOs is the number of times the MVS channel scheduler was entered;

- Total count of data, in bytes, transferred to and from the terminal since ULOG ON;
- Total count of data in bytes, sent via message switching or printout spooling since ULOG ON;
- Total number of transactions for the session (number of times in the thread) including the use of the rollout function.

The Checkpoint Record

This record is written once every *n* minutes for each user logged onto Com-plete, unless a program termination record has been written in the last *n* minutes or no activity has occurred for the user ID in the last *n* minutes. The checkpoint record contains the same information as the program termination record.

The checkpoint interval *n* is an installation variable defined at Com-plete initialization time using the SMFsysparm.

COM-PASS also produces a checkpoint record whenever a program is suspended. The checkpoint record can be examined at offset 63 (decimal). A "blank" (X'40') indicates a normal checkpoint. An "S" (X'E2') indicates a COM-PASS suspend record.

The ULOG OFF Record

This record is written after a user issues a ULOG OFF. The ULOG OFF record contains the same information as the program termination record.

The User Record

This record contains the standard SMF prefix followed by a freely definable area available for your use. The first half-word of this area must contain the length of this user-specific area (incl. length field).

In order to write the user SMF record, you must call (via TCALL) entry point ALSRBSMF (in COMREG) with the following registers:

Register	
0	User SMF record (will be appended to common part before being written).
1	4 (user record indicator).
2	COMREG.
4	THCB.
5	TIB.
13	18-F/W save area.
14	Return address.
15	Entry point.

SMF Record Contents

This section describes the fields and their contents that are written in the Com-plete SMF records. The offsets of the fields in the record can be determined from the DSECT produced by the CCCSMF copybook.

SMF Record Common Portion

The following fields are found in all SMF records written by Com-plete.

Field Name	Description of Contents
SFCRDWL	Contains the length of this record in binary format.
SFCSYSI	Contains the system ID. The following equates are available in the CCCSMF copybook and can be compared to the contents of this field to determine the operating system.
	SFCMVS MVS SYSTEM
	SFCVSE VSE SYSTEM
	SFCESA MVS/ESA SYSTEM
SFCTYPE	SMF record number as provided in the Com-plete sysparms in binary format.
SFCTIME	Time the record was written as a binary value representing the number 1/100ths of seconds since the start of the day.
SFCDATE	As a packed decimal value representing the date in Julian format when the record was written (i.e. 00YYDDDS).
SFCESYSI	Contains the 4 byte character SMF system ID.
SFCUSERI	Contains the 8 byte Com-plete user ID of the user for whom the record was generated.
SFCACCT	Contains the twelve byte account number related to the user.
SFCAUTH	Contains the user's authorisation code as a binary number.
SFCCNTL	Contains an indicator as to the control status of the user. The following values are possible:
	C Control user
	N Noncontrol user
SFCSNEWF	Contains an indicator to determine whether the record is of the newer type written by Com-plete 4.5 and subsequent releases or not. When it contains x'ff', the record has been written by Com-plete 4.5 or a subsequent release. Otherwise it has been written by a pre 4.5 release.
SFCSMC	Contains the class codes in numeric format (i.e. 1, 2 etc.) which this user is allowed to send.

Field Name	Description of Contents
SFCRMC	Contains the class codes in numeric format (i.e. 1, 2, etc.) which this user is allowed to receive.
SFCCTYPE	Contains an indicator as to the type of Com-plete SMF record being dealt with. The CCCSMF copybook contains the following equates which can be used with this field to determine the record type.
	SFCTLOGO Logon type record
	SFCTTERM Program termination
	SFCTCHKP Check point type record
	SFCTSUSP Suspend type record
	SFCTLOGF Logoff type record
	SFCTUSER User SMF record.
SFCUSER	Contains an indicator as to the user type. The CCCSMF copybook contains the following equates which can be used to determine the user type.
	SFCMATT Attached user
	SFCMSCHD Scheduled user
	SFCMNORM Logged on or "normal" user
	SFCLUNAM Contains the terminal name of the terminal on which the user was running. For VTAM terminals, this is the VTAM LU name, for batch users, this is the name of the batch job.
	SFCPATCH The patch character for the Com-plete where this user was running.
	SFCVRM Contains the Version, release and SM level respectively of the Com-plete from which the record was written.
	SFCTID Contains the terminal number in binary of the terminal on which this user was running.

SMF Record Statistics Portion

The following fields are only found in the program termination, checkpoint and logoff records

Field Name	Description of Contents
SFTPROG	Contains the eight byte alpha-numeric name of the last program the user ran. This is not applicable in the logoff record.

Field Name	Description of Contents			
SFTPRIV	Contains the privileged status of the program (except in the logoff record). It can contain the following values.			
	<table> <tr> <td>P</td> <td>Program was privileged</td> </tr> <tr> <td>N</td> <td>Program was not privileged</td> </tr> </table>	P	Program was privileged	N
P	Program was privileged			
N	Program was not privileged			
SFTABEND	Contains the termination status of the program. It can have the following values.			
	A Program abended			
	N Normal termination			
S Program suspended in COM-PASS				
SFTTHRN	Contains the number of the thread in binary format where the program ran.			
SFTTRANS	Contains the number of times the user has been marked as being eligible to roll out. This is not necessarily the number of user rollouts as a user program is only rolled out if another user program needs to use the thread.			
SFTCPU	Contains the total amount of CPU used by the user in 1/100ths of seconds.			
SFTCPU TU	Contains the total amount of CPU used by the user in machine timer units.			
SFTELAP	Contains the total length of time the user spent in a Com-plete thread 1/100ths of seconds.			
SFTEXCP	Contains the total number of I/Os issued by or on behalf of the user.			
SFTDATA	Contains the total amount of data in bytes which was sent to and received from the terminal.			
SFTMESG	Contains the total amount of data in bytes which was spooled by this user using the Com-plete printout spooling facilities.			
SFTLFTIM	Contains the time that this program was started either directly via Com-plete or via a FETCH. It is a binary number which represents the number of 1/100ths of seconds since the start of the day (i.e. time '00:00').			
SFTUSER	This field is copied from the UAB field available to users called UABUSER.			
SFTTRAN#	Contains the number assign to this program when it was started. When a program is started in Com-plete via a Com-plete or using a fetch, it is assigned a unique number which starts with one for the first program started.			

Field Name	Description of Contents
SFTRTIME	This contains the total of all response times for this user. A "response time" is the time between when Com-plete has associated an incoming user data stream with a terminal (i.e. when the user presses ENTER) and the time when Com-plete determines that the output has been provided back to the user.
SFTRTIM#	This contains a count of the number of "response times" which Com-plete has seen for this user. This number is basically the number of times this user has pressed ENTER at the terminal.
SFTADAC	Contains the total number of Adabas calls that this user has issued.
SFTADATR	Contains the total number of times a screen interaction has included adabas calls. This figure will always be equal or less to SFTRTIM# depending on whether the user runs applications that generate no Adabas calls between screen I/Os.
SFTOPCNT	Contains the total number of Com-plete operations which this user has issued. A Com-plete operation is a MCALL or other operating system function which Com-plete has satisfied internally. An example of an operating system function which Com-plete will satisfy is a LOAD request from a user application program.
SFTADAET	Contains the total amount of time that the user has been within the Adabas link routine ADALCO. This represents the total elapse time between when the call was issued to Adabas and when the response was received back from Adabas. This number represents the time in milli seconds (1/1000ths).
SFTADATD	Contains the total amount of time the user has spent in Adabas. This field is accumulated from the ACBDUR field in the Adabas ACB for each Adabas call issued. Please refer to your Adabas documentation for more information on what the ACBDUR field represents and its format.
SFTTQTIM	Contains the total amount of time this user spent on the Com-plete ready-to-run queue in milli seconds (1/1000ths).
SFTTQCNT	Contains the number of times that this user was enqueued to the Com-plete ready-to-run queue.

Modifications to Com-plete Modules

This chapter summarizes maintenance procedures for Com-plete modules under the following headings:

- Overview
 - Link Editing Com-plete Modules
 - Link Edit Return Codes
 - Com-plete Support Issues
-

Overview

Alterations to the Com-plete modules must be performed in a consistent and documented manner to ensure the stability of the product.

Com-plete modules may need to be altered with fixes supplied by SOFTWARE AG. In addition, system-wide modifications can be implemented to customize the operation of the Com-plete nucleus and its utility programs to the needs of the installation. The following types of customizations are acceptable:

- Using the APPLYMOD sysparm that alters the functioning of Com-plete;
- Adding user exit routines to Com-plete and utility programs for security and customization;
- Modification of maps to present terminal users with an organization's specific screen.

Link Editing Com-plete Modules

As all user exits are now loaded as standard, there should be no reason to relink Com-plete modules. However, for reasons of compatibility, exits that were previously documented as being linked with Com-plete can still be linked. This facility will be dropped in a future release of Com-plete. If a Com-plete module needs to be linked with a user-written module, the resultant load module must be stored in a user data set so that the original is still available. If a problem is not reproduceable with the supplied Com-plete modules, then it will not be accepted as a product problem.

In the case of maintenance being applied to modules that must be linked, you are recommended to copy the applicable module from the Com-plete distribution library to a Com-plete zap library, zap it there and relink it to the zap library. When you have verified that the zap has gone in correctly, the applicable modules can be copied back to the distribution library. On the other hand, you may prefer not to alter the distribution library and leave the zapped modules on the zap library. However, in this case, you must ensure that the module(s) on the zap library are not overwritten, otherwise the zap update is lost.

The linked edits for the nucleus and utilities for MVS and FACOM is found in the member \$LINKxx, where xx stands for MVS or F4 as appropriate to your operating system.

Note:

Newly linked utilities may or may not become available in the system immediately due to a fixed number of BLDL/LOAD list entries held for the most recently used modules while Com-plete is active. Use the ULIB REFRESH command to ensure a newly linked utility becomes available immediately.

Link Edit Return Codes

In most cases, the link-edits for Com-plete should have a return code of 00, otherwise the results of executing the newly linked module are unpredictable. The following are the link edits that can be expected to finish with a return code of 04. Anything higher is again an error and execution of this module causes unpredictable results.

TUDUMP
TUMSUTIL
TUSACAPT
TUSDUTIL
UED
UPWD

Com-plete Support Issues

SAGSIS Problem System

Software AG provides a data base all customer requests and reported problems. Access to this data base is made via the SAGSIS Problem System to which all affiliates online to the Reston or Darmstadt data centres have access.

SAGSIS has two different entities, the "request" and the "problem". When a customer initially contacts the support centre, generally a SAGSIS request is entered depending on the nature of the contact. For example, a question from a customer already found in SAGSIS does not need to be entered again. If a Request is set up, the customer will receive a request number. This is a unique reference number for this particular customer contact and should be used to identify any materials sent to SAG in relation to this contact.

In some cases, the request will be found to be a software problem. In this case, a problem entry is set up to describe the actual problem and the solution to the problem. This will have a unique problem number. However, the only numbers that concern the customer are their individual request numbers. All data related to the request is then linked to this number.

It is in the interest of all parties concerned to have as much data in this data base as possible because the more data there is, the more solutions can be found quickly in the data base. For this reason, we strongly recommend that all customer problems are registered through the support centres and, where applicable, that customers always ensure that they receive a request number for the "problem" report.

Problem Reporting

To ensure a fast turnaround time when reporting a problem, the following information must be provided, depending on the type of problem.

1. Version, release and SM level of the Com-plete in which the problem occurred;
2. Type and level of operating system under which the Com-plete is running;
3. Version, release and SM level of products related to the problem (eg. Natural, Adabas);

4. Message numbers where applicable;
5. System log for a period of time before the actual event;
6. Sequence of actions used to cause the problem if available;
7. Name and offset in module where an abend occurred if applicable.

On the basis of the above information, a search can be made within the system. If this does not lead to a solution, the above information must be supplemented with the following.

1. Dump resulting from the problem if applicable;
2. Output from the job where the problem occurred;
3. Any other information that may be requested by the support personnel.

Where practical information can be sent on paper (for example, thread dumps), this is recommended. However, for larger dumps, the dump should be formatted and contain all the relevant control blocks and storage areas. Depending on the nature of the problem, only the actual Com-plete areas may be necessary. However, you should consult with support personnel as to what they require.

Com-plete Problem Solutions

When a problem is identified and a correction of some nature is required, the correction may take any of the following formats.

1. A zap which must be applied to various Com-plete modules;
2. A source update which indicates the source member on the distribution to be changed and what should be changed there;
3. An update tape with updated modules and source members;
4. A new System Maintenance level of the product;
5. An acceptance of the request as a change enhancement request to be included in a future release of Com-plete.

The normal situation is a simple zap which is discussed in the section **Com-plete Fixes** below. Situations where a source update is required can sometimes be corrected with a description of what needs to be changed. However, if the changes are so great that the possibility of errors exists, the updated source will be distributed by tape.

In the case where a correction is so complex that it cannot be zapped, if the problem is not a major one, the request will be treated as a change enhancement request. If the problem causes major problems which are fixed in a system maintenance level, the user will be requested to go to that particular level. If it is not fixed, the correction will be distributed on tape with replacement modules and source members as necessary.

Com-plete Fixes

From Com-plete 4.5 on, a new zap system has been in use to track zaps on a very simple basis and present a standard format for the zap when it is sent to customers. It is also used to avoid a customer receiving all operating system portions of the zap. In the future, you should only receive the zap data that applies to your operating system.

The system also provides for the orderly correcting of the original problem in source and the creating of SM levels. This will lead to more stable SMs in the future and will avoid zaps being left out of the newly sourced level.

The zap name/number has the following format:

CPvrnnn

where

CP	is constant for Com-plete
vr	is the version/release that the zap applies to
nnn	A unique number for the zap

Only one TLFIX module is now used in Com-plete to record all applied fixes. This offset in the module relating to the number of the fix is, as before, zapped to indicate that the zap is applied. However, the information zapped in is different depending on the status of the zap. If a zap has been applied to an SM level, the the appropriate SM level number is zapped into the applicable location so that you can see what SM level a zap belongs to, or on which SM level tape the zap can be expected to appear. If the zap does not have a currently assigned SM level, then the value "ff" is set to simply indicate that the zap is applied. The ZO function of UUTIL then uses this information to give a more informative view of what zaps are applied.

The Zap Format

Zaps are distributed in the following format. Some of the information may be of use to you and some is not. Each heading is described following this example.

```
* Identification:  a
* Problem:       b
* Product:       c
* Creator:       d
* Creation date:  e
* Zap Status 2:  g
* Release date:  h
* OPSYS:        i
* Description    j
*
*
NAME TLFIX TLFIX  LOADLIB(TLFIX)
VER nnnn 00  IS ZAP APPLIED ?   (k)
REP nnnn FF  FLAG AS APPLIED, NO SM LEVEL ASSIGNED (k)
REP nnnn ss  FLAG AS APPLIED, SM LEVEL ss          (k)
*
NAME mmmmmmmmm ccccccc LOADLIB(1111111,11111112,...,1111111n) (1)
```

```

VER xxxx xxxx,xxxx,xxxx      code          (m)
REP xxxx xxxx,xxxx,xxxx yyyy code          (m)
*
IDRDATA a

```

code	Description				
a	This is the number identifying the zap.				
b	This is the problem number associated with the zap. Please note that this is an internal problem number, possibly connected to your request; however, it bears no relation to any request numbers that you may have.				
c	This is the product name, this is the constant COMPLETE.				
d	This is the user ID of the SAG person who created then zap.				
e	This is the date and time the zap was created.				
g	This is the status 2 of the zap and gives additional information about the zap. It is not always specified, in which case it does not appear. However, when it is there, it is followed by a short explanation as to why it has that status 2. When this is the case, you should consult support personnel as to whether it is safe to apply the zap. The following are the possible status 2 values.				
	<table border="0"> <tr> <td style="padding-right: 20px;">Error</td> <td>This indicates that the zap has caused problems. In general, zaps with this status 2 must not be applied.</td> </tr> <tr> <td>Held</td> <td>This is more information as to what the zap does. It generally relates to effects noticed after the zap is created.</td> </tr> </table>	Error	This indicates that the zap has caused problems. In general, zaps with this status 2 must not be applied.	Held	This is more information as to what the zap does. It generally relates to effects noticed after the zap is created.
Error	This indicates that the zap has caused problems. In general, zaps with this status 2 must not be applied.				
Held	This is more information as to what the zap does. It generally relates to effects noticed after the zap is created.				
h	The release date is the date and time when the zap is considered eligible for general release. Normally you only receive zaps of this nature. However, in exceptional circumstances or when you are testing a zap for a problem, a zap will be sent to a customer and the string not QA'ed appears in this field.				
i	This is the operating system that the zap relates to. The zap statements are only printed and valid for this operating system. The possible operating systems are listed below. Some operating systems are "generic", which indicates that the zap is good for the corresponding "set". *				
j	This is a short description of what the zap fixes. A more detailed description can be found in the SAGSIS problem entry.				
k	"nnnn" here is the offset in the TLFIX module that reflects the zap number. The REP line will be one or the other depending on the status of the zap as stated previously.				

code	Description
l	"nnnnnnn" is the name of the module, "ccccccc" is the CSECT name and "lllllll" to "llllllln" is the name of the load module(s) into which this module is linked. If the module name is the same as the load module name, no linking is necessary. If the module name is not the same as the load module name, the linkedit for the load module must be taken and the module relinked as explained at the beginning of this chapter.
m	The various lines of VERs and REPs then follow, the comments to the right are simply for documentation purposes.
n	The zaps for VSE are shipped in MSHP format.

*

ALL - Good for all operating systems and levels

MVS - MVS/XA, MVS/ESA, OS/390

OS - F4, VOS, MVS/ESA, OS/390

VSE - VSE/ESA 2.1 and higher

Com-plete Maintenance Updates

As zaps are accepted, they will be applied to a SM+1 frozen library, that is, if the current release is 5.1.1, the zaps will be applied to a 5.1.2 frozen library. System Maintenance levels of Com-plete are planned to be released every three to four months. When these SMs are distributed, a fourth data set will exist on the installation tape with the various zaps applied to the load library in source. If you do not wish to install the SM tape, the zaps are then available in source to be applied on an adhoc basis.

Com-plete Capture Processing

Com-plete capture processing is a mechanism for the "capturing" and saving of data of any type for later use. The data is captured to so-called CAPTURE data sets which must be allocated and initialized before Com-plete is initialized. As well as capturing system data, such as the trace records, installation-specific data can also be captured.

This chapter covers the following topics:

- Capture Data Sets
 - Captured Data
 - Capture Records Processing
-

Capture Data Sets

The Capture data sets are VSAM data sets allocated as specified in the section **Com-plete Files and Associated User Files** in this documentation. They must then be included in the Com-plete job control with the DDname/DLBL name CAPTUR n , where n is a number from 1 to 9. At initialization, capture processing either uses the number of Capture data sets as specified in the CAPTURE sysparm, or determines the number of data sets specified in the job control dynamically.

Note:

To disable Capture, you must specify SUBSYS-IGNORE=CAPTURE in the Com-plete sysparms.

By default, the capture data sets are non-reusable. This means that when they are filled, they cannot be used again until they have been copied and/or reset by the TUSACAPT utility. To enable the capture subsystem to reuse the data sets when they are filled, specify CAPTURE=(n ,REUSE) in the sysparms.

After initialization, Capture opens the first available data set and uses it to capture data. The capture data currently in use is always clearly visible from the COMCA... messages issued to the operator console. When a data set fills, the capture subsystem informs the operator and attempts to find another data set that can be used. If the REUSE option is specified on the CAPTURE sysparm, the next data set is selected, opened and used (in round robin fashion, from CAPTUR1 to CAPTUR9), regardless of whether it already contains data or not.

If the REUSE option is not specified, and the current data set fills, the next data set is checked for data. If data exists, the following data set is checked, and so on, until a data set without data is found. In the event that all data sets contain capture data, a message goes out to the operator, informing him/her to clear a data set for use, otherwise capture processing cannot continue. Com-plete then waits until the operator replies that one or more data sets have been cleared before checking again for an 'empty' data set. For this reason, it is recommended that you build a procedure to unload and initialize Capture data sets as soon as they are filled.

Captured Data

When a Capture data set is opened, either for the first time or after it has been unloaded and initialized, a header record is written as the first record to the Capture data set. This contains details of times, dates, software levels etc. of when the data set was opened. The exact contents and format of this header record can be found in the CAPLAB macro supplied on the distributed Com-plete source data set.

Following this header, system and/or user capture records can be found. Each record has a prefix to identify the type and format of the following record along with the date and time it was written, followed by the actual data itself. Refer to the CCCAPT copy book supplied on the Com-plete distributed source for details of the layout of this area.

Capture Records Processing

Com-plete provides no utilities to process these records, as the contents of the records is purely up to the user. In the case of system records, a utility will be supplied in a future release to format the Com-plete trace records that are written.

Com-plete Servers

This chapter covers the following topics:

- Overview
 - Server Definition
 - Server Initialization
 - Server Termination
 - Server Command Interface
 - Server Invocation
 - Server DSECTs
 - Server Request Routine
-

Overview

The modifications to Com-plete required for the implementation of the DB2 interface has resulted in the introduction of a new independent server facility to Com-plete. This facility enables Com-plete to support products which require their own subtask(s) and/or special invocation during startup and termination.

In addition to the DB2 interface, other SOFTWARE AG products such as the Natural Bufferpool Manager, the SAG EDITOR and the Com-plete JES and CONSOLE interfaces have been adapted to use this facility.

3rd party products such as TableBase (Data Connetics) and TABEX (BOI) can also use this feature.

In general, the functions to be provided by user-defined servers have no restrictions. Com-plete provides a basic framework of facilities, consisting of the following:

- Definition
- Initialization
- Termination
- Command Interface
- Invocation

These are described in more detail in the following sections. This chapter also illustrates server DSECTs, and provides information on the server request routine.

Server Definition

You can define servers using the SERVER sysparm:

```
SERVER=(serv-id,init-mod,p1,p2 ... pn)
```

where:

serv-id	is the ID for this server (1-8 characters).
init-mod	is the name of the initialization/termination routine.
p1,p2...pn	are parameters to be passed to the initialization routines.

Com-plete builds a server directory dynamically and provides a Server Directory Entry (SDE) for each server. Note that user-defined servers must be defined *after* the Com-plete servers.

The layout of the SDE is illustrated in the section **Server DSECTs**. It is mapped by the macro

```
CMSRVD DDIR,LIST=YES
```

provided on the distributed source library.

Server Main Routine

The server initialization/termination routine specified in the SERVER sysparm entry is loaded dynamically at Com-plete startup time and called in the key of Com-plete. In MVS and VSE environments, the program is called according to the AMODE and RMODE options from the linkage editor and must return with the AMODE indicated in RE on entry (BSM instruction).

The main routine executes under control of the main Com-plete task (OC).

All operating system functions are available, except Com-plete operations (MCALLs), which are not allowed.

Registers upon entry (all calls):

R1	parameter list pointer (see individual function).
R2	COMREG
RD	18F save area
RE	return address (high order bit contains AMODE of caller)
RF	entry point

Server Initialization

The server initialization takes place after activation of any Com-plete subtasks (TAM, MSGPO, THREADn). If the user-defined server attaches any further subtask(s), you must consider the priority carefully, since the default from the ATTACH SVC will put this subtask immediately behind OC, meaning that the later activation of TAM, MSGPO etc. will put these Com-plete tasks behind the server task(s). A value of minus 2 (-2) or less for DPMOD on the ATTACH is recommended.

Parameter list for initialization:

P1	address of function code "INIT".
P2	address of the Server Directory Entry.
P3	address of the parameter string from the SERVER sysparm.

The parameter string from the SERVER sysparm definitions is constructed in the following way:

(LL(P1.....,Pn))

where:

LL	2 bytes containing the total length of all parameters (not counting LL field itself).
P1,..Pn	parameter value(s).

Example:

```
SERVER=(TEST,TLINTEST,aaa,bbb,r=17)
```

The data pointed to by R1 will look like this:

```
x'000C',C'aaa,bbb,r=17'
```

After successful completion of the server initialization, the fields SRVEMCB and SRVESERV must be set in the SDE. The meaning of these fields is as follows:

SRVEMCB	The content and meaning of this field is up to the individual server. However, the initialization routine must ensure that this field contains a non-zero value, otherwise the server is considered to be not active.
SRVESERV	Contains the entry-point for server calls via the ALLOCATE/DEALLOC SERVER function (see below). The high-order bit of this address is used to set the AMODE in XA/ESA environments.

Server Termination

Since the termination process is basically a reverse initialization, this request is also passed to the main-routine (see above).

Termination calls are triggered either by Com-plete termination or using the operator command

```
SERV TERM,serv-id
```

The server routine is responsible to free all obtained storage, DETACH the activated subtask(s) and to CLOSE any open datasets.

Parameter list for termination:

P1	address of function code "TERM"
P2	address of the Server Directory Entry

The server routine has to ensure that the SRVEMCB field is set to zero.

Server Command Interface

Various requests can be passed to the individual server using the Com-plete operator command

```
SERV serv-id,request data
```

Parameter list for operator interface:

P1	address of function code "CMND"
P2	address of the Server Directory Entry
P3	address of request data string from the command (for format, see the section on server initialization above.)

Server Invocation

The invocation of a user-defined server can be achieved in 2 different ways.

1. The server directory entry for a given SERVER-ID can be found by using the following macro:

```
CMSRVD QUERY,NAME='serv-id'
```

This function needs addressability to COMREG (any register) and will return the address of the directory entry in R1 if this server is up (R15=0), otherwise R15 will be non-zero and the contents of R1 are undefined. For a description of the server directory entry SDE see the section **Server DSECTs**.

The value in the SRVEMCB field can be used to obtain the address of the specific service routine.

2. If a user-defined server requires something like session control, you can use the ALLOCATE and DEALLOCATE functions of the CMSRVD macro. These functions branch to the SRVESERV entry point of the user-defined service routine. The expected functionality of this routine is described in the section **Server Request Routine below**.

This alternative ensures that Com-plete performs an invocation in cases where the application ends (normal or abnormal) and still has a pending session with a server.

To acquire a session with a server use the macro:

```
CMSRVD ALLOCATE,NAME='serv-id'
```

After successful execution (RF=0), this macro returns the address of the RQE in R1 (see the section **Server DSECTs**).

R15 > 0 indicates that the request could not be satisfied and the contents of R1 are unpredictable.

To free a session with a server, use

```
CMSRVD DEALLO,NAME='serv-id', AREA=
```

The AREA keyword specifies a register or storage location which contains the address of the RQE provided by the ALLOCATE function.

The corresponding branch interface function is

```
SERVER,(retcode,serv-id,function,area)
```

where:

retcode	required A fullword where Com-plete places the returncode upon completion of the request.
serv-id	required An eight-byte character field containing the servername (as stored in SRVENAME)
function	required An eight-byte character field containing QUERY, ALLOCATE or DEALLO
area	required only for function DEALLO A fullword containing the address of the RQE provided by the ALLOCATE function

More about the requirements of the branch interface can be found in **Application Programming Interface** in the Application Programming documentation.

Server DSECTs

The server directory entry (SDE) is mapped in the DSRVE DSECT, which is generated by the following macro:

```
CMSRVD DDIR,LIST=YES|NO
```

```
DSRVE          DSECT,          SERVER ENTRY
SRVENAME       DS CL8          NAME
SRVEINIT       DS CL8          NAME INIT ROUTINE
SRVEINEP       DS A            EP INIT/TERM ROUTINE
```

SERVER DIRECTORY available for server

```
SRVEMCB        DS A            A(SERVER CONTROL BLOCK)
                                ZERO INDICATES NOT UP
SRVESERV       DS A            A(SERVICE ROUTINE)
                                DS A            ** RESERVED **
SRVELEN#       EQU *-DSRVE     LEN OF ENTRY
```

The server request entry (RQE) is mapped in the DSRVE DSECT, which is generated by the following macro:

```
CMSRVD DRQE,LIST=YES|NO
```

```
DSRVR          DSECT,
SRVRNAME       DS          CL8      ID
SRVRLEN DS     F          len of control block
SRVRNSRV       DS          A          A(next request element)
                                     maintained by Com-plete

SRVRREQE       DS          A          A(SERVICE ROUTINE)
SRVRSVFE       DS          A          A(entry for ALLOCATE, DEALLO,
                                     and EOJ function)
                                     DS          2A          ** RESERVED **
```

The default for the LIST keyword is the setting of &CCPRNT (CCGLOBS).

Server Request Routine

The server request routine is provided in the field SRVESERV of the SDE. The routine is invoked using ALLOCATE, DEALLOC requests via the CMSRVD macro and during EOJ processing for a user session.

Registers upon entry (all calls):

R1	server directory entry (SDE)
R2	COMREG
R7	address of option (ALLOCATE, DEALLOC, EOJ)
R8	AREA from DEALLOC
RD	18F save area
RE	return addr (high order bit contains AMODE of caller)
RF	entry point

ALLOCATE Request

The server must allocate a RQE outside the thread. Other server specific functions may also be performed during ALLOCATE processing. If the request was successfully processed R15 must be set to 0 and R1 must contain the address of the RQE.

DEALLOCATE Request

Deallocate processing must be used to free all user-related resources or whatever may be necessary for the server. After return from the server request routine, Com-plete will remove the RQE from its internal chain. This will prevent any cleanup call at session end.

EOJ Request

If an active server session is pending at EOJ processing for a transaction, Com-plete invokes the server request routine for cleanup processing (SRVRSVFE).

The option pointed to by R7 on entry indicates the type of transaction end:

EOJNORM	normal end of transaction
EOJABEND	transaction abended

Software Interfaces

This part of the Com-plete System Programming documentation discusses the considerations when using Com-plete in various software environments.

This information is organized under the following headings:

- ACCESS
- Adabas
- Application Programming Interface
- Batch
- VTAM
- APPC Interface
- CICS / Com-plete Transaction Routing
- LIBRARIAN (MVS Only)
- VSE LIBR Service
- Using VSAM with Com-plete
- GDDM
- PANVALET (MVS only)
- Job Entry Subsystem (JES) Interface Modules
- UDS VSAM SERVICES (MVS Only)
- UDVS VSAM SERVICES (VSE Only)
- UDS VSAM SERVICES (MSP/FACOM Only)
- Security Systems
- The DB2 Interface
- Natural
- CA-DYNAM from Computer Associates (VSE only)
- IBM Language Environment Considerations
- File Transfer

ACCESS

ACCESS is the term given to the Com-plete access method: communication is performed via the Adabas router interface.

This is used, for example, to provide communication with other TP-monitors in the Adabas TPF environment. It is also used in Com-plete Batch support to provide the basic communication between the Batch job and the Com-plete Batch interface utility UBATCH.

Communication with the Adabas router is provided by modules supplied on the Adabas load library. These modules are loaded during initialization from the Adabas load library, which must therefore be in the COMPINIT concatenation in the Com-plete start up procedure.

Com-plete issues the appropriate calls to sign onto and initiate communication with the Adabas router using the parameters supplied during Com-plete initialization.

Sysparm Considerations

Sysparm ACCESS-NCQE defines the number of command queue elements which are allocated by the Adabas router. This restricts the maximum number of concurrent Com-plete transactions. Note that this is not the same as the number of logged on users: this is normally higher and is only restricted by the availability of TIBTAB entries.

Sysparm ACCESS-NABS defines the number of attached buffers which are allocated by the Adabas router for cross-memory services. The Adabas router acquires storage for the attached buffers during ACCESS initialization, the size of this acquired area is approximately (NABS * 4 Kbytes). This restricts the amount of data which can be transmitted. Adabas TPF requires approximately 7K of attached buffers per active transaction (see ACCESS-NCQEs).

Sysparm ACCESS-ID defines the (pseudo-)data base ID with which the server Com-plete signs on to the Adabas V5 router.

Sysparm ACCESS-SVC defines the router SVC number. Note that this SVC need not be the same as for normal data base communication.

Adabas

The *Com-plete Application Programmer's documentation* describes the call interface to Adabas. This call interface provided by Com-plete deals exclusively with Com-plete timing, rolling and statistics. The Adabas linkage is provided by separate ADALNK modules.

This chapter covers the following topics:

- General Usage
 - ADALNK Features
-

General Usage

All Adabas calls are issued in the key of the thread itself. As there is a requirement to sometimes roll out a user over an Adabas call, the Adabas interface requires that it can get its UB area and a certain amount of its workarea outside of the thread and in the same key as the thread. For this reason, an Adabas buffer pool is built with subpools of the same size but in different storage protect keys.

The size of the Adabas workarea is dependent on the various lengths set in the ADALCO module when it is assembled. It is possible to have a user length and a Review length included in the calculation for the Adabas workarea in this way. The total amount of the Adabas workarea should be rounded up to a good binary figure such as 512 or 1024. This is because the buffer subpools must be page aligned and without a value such as one of these, storage will be wasted.

If no Adabas buffer pool is specified by the user, Com-plete allocates a buffer pool with a subpool in each user storage protect key possible with 8k available in each. The number of Adabas work areas that can be allocated from each will be dependent on it's total length. It is not possible for Com-plete to make a more intelligent guess than this as their usage totally depends on the number of Adabas calls which will be issued concurrently in a given storage protect key.

It is recommended that users calculate their Adabas usage and build their Adabas buffer pool accordingly. You must build a buffer pool with a subpool available in every key in which an Adabas call will be issued. If an Adabas call is issued in a key for which no subpool exists, the subpool will be created but this in itself is an additional overhead. Software AG recommend that a subpool be created for each key in which an Adabas call can be issued in the Com-plete region.

You must now determine how many Adabas work areas will be required at the same time in each subpool. Software AG recommend that the resultant figure be increased by 20% to allow for peaks and future use. If the subpool fills, it will expand if the virtual storage is available but again, this is an overhead. If the subpools are well allocated and their usage monitored closely, it should be possible to increase or decrease the specification as required.

The Adabas work areas will be allocated above the line when the following conditions are satisfied. If one of the following is not satisfied, the Adabas buffer subpools will be allocated below the line.

ADALCO must be linked with AMODE=31

If ULOPADAB exists it must be linked with AMODE=31

There must be no Adabas version 4 usage in the system

ADALNK Features

The V5 ADALNK features of user exits and data passing are supported by the Com-plete interface:

- **User exits**

If required, you can link the UEXITB and UEXITA routines with the supplied ADALNK module. The ADALNK routine is called in the AMODE in which the MCALL Adabas was issued, this also applies to UEXITB and UEXITA. You must ensure that the return is made in the correct mode, that is, in the mode in which he was entered. Please note that parameters passed from Natural can reside above the 16 MB line.

- **Data passing**

The ADALNK module is also supplied in source so that you can re-assemble the module providing a value for LUINFO, the UB extension length. This length is added to the basic length of the workarea (approx. 512 bytes) and the total is the size of the area obtained by the Com-plete nucleus from the buffer pool. This means that careful adjustment of the BUFFERPOOL sysparm may be necessary to prevent inefficient use of storage.

A full description of these features is contained in the Adabas documentation.

Application Programming Interface

The application programming interface is now what is termed "storage key sensitive". This means that the API routines determine the key of the storage to which they are going to write and switch their PSW key to that key prior to writing to the storage. This avoids the overhead of changing the entire thread to Com-plete's key during an API call and back again once the call has been processed. It also means that the path length through an API routine is exactly the same if the thread is defined with Com-plete's storage protect key or a different protect key i.e. using the storage protection feature should be no more expensive for most applications.

Note that some emulated operating system SVCs cause the key of the thread to be changed due to internal operating system requirements. For this reason, operating system specific functions should be avoided whenever possible and the appropriate Com-plete API function used instead.

Due to the requirement to support SVC entry to the nucleus, there is a slight additional overhead required in the thread. This is primarily an 18F savearea which is used by the internal bridging mechanism to enter the standard API routines. Where COLINK, or OS type link requests are used, an additional 18F savearea is acquired per link request which is freed on return from the link.

Batch

Batch applications which require Com-plete services use ACCESS to communicate with the target Com-plete system. As the mechanism used to communicate with the Com-plete region is the Adabas router mechanism, this method of batch support can take advantage of the networking features provided by the Adabas router mechanism.

As each server Com-plete signs on to the router with a different data base ID (which is also unique within the network), several Com-plete regions can offer batch support, allowing a distribution of the batch workload between multiple address spaces.

For further information on how to install ACCESS, see the description of the ACCESS interface. For further information on how to install Natural Batch, see the description of the Natural interface.

This chapter covers the following topics:

- Running Batch Programs
-

Running Batch Programs

1. Batch applications must be linked to the module COMPBTCH supplied on the Com-plete load library, which contains entry points for all Com-plete functions allowed from a batch environment.
2. COMPBTCH loads modules TUBATCH and TUBCONN to perform the required function. These load modules must therefore be contained in the STEPLIB concatenation of the batch job.
3. On the first call, TUBATCH will attempt to establish communication with the target Com-plete. To do this, TUBATCH requires two pieces of information: the NODEID of the target node and the SVC number of the Network Router (Adabas SVC) on which this node is established.

This information can be obtained in one of the following ways:

- ACSTAB.
TUBATCH attempts to load a table ACSTAB and, if found, will search this for the entry named BATCH, the values for NODEID and SVC are then taken from this entry.
A sample ACSTAB is delivered on the Com-plete source dataset. This member must be modified to suit the installation requirements, assembled and linked to an appropriate data set during installation of Com-plete. This library can then be included in the STEPLIB concatenation of batch jobs.
- COMBTCH JCL statement.
In addition to the above procedure, TUBATCH will always attempt to find the NODEID and SVC from JCL DD card COMBTCH.

```
COMBTCH DD DSN=NODEnnnn.SVCsss,DISP=.....
or: COMBTCH DD DSN=NODnnnnn.SVCsss,DISP=.....
```

The values obtained will overwrite the values contained in the ACSTAB to allow individual jobs to override values specified in the ACSTAB.

Note:

The value of NODEID in the ACSTAB may be 1 - 65535. To allow for 5-digit values, NODnnnnn is also supported

- a combination of both. This method allows you to define multiple Com-pletes in one ACSTAB. You define ACSTAB entries using any names you like (e.g., jobname or installation ID or whatever) instead of or along with the "BATCH" entry. To select an entry, use a COMBTCH DD statement like this:

```
COMBTCH DD DSN=[any_prefix.]TARGET.target[.any_suffix],DISP=...
```

where target is the entry name associated with the COMPLETE you want the BATCH job to connect to. If this notation is used and NODEnnnn and/or SVCsss are also specified, then the latter are ignored.

4. The batch interface modules supply a default user name of BATCHCOM at logon to the target system, except when the job is submitted in an environment which is protected by an external security system; in this case, the user ID under which the job is submitted (taken from the ACEE) is passed to the target system.

Note:

The target Com-plete node must be started with BATCH=YES specified in the system parameters.

VTAM

The VTAM interface is the Com-plete communication channel to the SNA network. It maps the terminal I/O requests to the SNA protocol corresponding to a specific device (LU) type. LU0, LU1, LU2, LU3 and LU6.2 protocols are supported. On activating the VTAM Interface, Com-plete becomes a VTAM application LU known in the network by its ACBNAME.

The VTAM interface is described under the following headings:

- Defining and Activating the VTAM Application
 - Generic Resource names
 - LOGMODES
-

Defining and Activating the VTAM Application

Before the VTAM interface can be activated, the application must be defined (in *SYS1.VTAMLST*) and activated (*VARY NET ACT,ID=acbname*):

```
name  APPL  APPC=YES/NO
      ,AUTH=(ACQ,PASS)
      ,ACBNAME=acbname
      ,PARSESS=YES
      ,MODETAB=mode_table
      ,SECACPT=ALREADYV
```

where:

name	specifies the network-unique name. If you do not code the ACBNAME parameter the network-unique name is used as ACBNAME. It must be identical to the name specified in the VTAMAPPL sysparm.
acbname	specifies the minor node name assigned to this application program. This name must be unique within the domain in which the application program resides. If you do not code this operand, the network-unique name (the name of the APPL definition statement) is used as the ACBNAME.
mode_table	specifies the name of a logon mode table to be used to associate each logon mode name with a set of session parameters. If you do not supply a logon mode table on the MODETAB operand, an IBM-supplied default logon mode table (ISTINCLM) is used. If you specify a table, both the table you specify and the default table are used. It is recommended to create a new mode table when new logon mode names are created or parameters on existing logmodes from the default table are changed.
APPC=YES	is required to activate the LU6.2 (APPC) support.
AUTH=ACQ	determines that Com-plete either the OPNDST macroinstruction with the ACQUIRE option or the SIMLOGON macroinstruction. (These macroinstructions enable Com-plete to initiate a session with a particular logical unit, e.g. a printer). If you code APPC=YES, this parameter defaults to ACQ and can be omitted. If you code APPC=YES and AUTH=NOACQ, VTAM supplies an override of ACQ and issues a warning message.
AUTH=PASS	is required if the PASS option of the ULOG utility is used, allowing VTAM to pass the session to another application. Otherwise the request will be rejected with RC=20 (X'14') FDBK2=94(X'5E').
PARSESS=YES	allows Com-plete to have multiple LU-LU sessions with the same session partner. PARSESS defaults to YES when APPC=YES. This option is required (explicitly or by default) when ULOG PASS is to be used or when Com-plete is used in conjunction with a session manager (as NET-PASS) in order to reduce the number of ACBs.
SECACPT=ALREADYV	Tells VTAM to build a BIND RESPONSE in order to allow "security"+"already verified" bits on incoming ATTACH requests.

Notes:

1. Check the NET-PASS sample exit NPEXIT07 on the distributed NET-PASS source library if NET-PASS is to provide the real LU-name of the terminal to Com-plete. The counterpart of this exit in Com-plete is activated automatically.
2. SONSCIP=NO (default) should always be in effect, otherwise Com-plete will not be able to detect session failures and perform the termination processing for that user. Further attempts to re-logon to Com-plete will be rejected (user already logged on).

3. See the description of APPLYMODs 29, 52, 53 and 69 for parameters affecting the VTAM Interface.
4. Mode modename must be defined in a mode table before Transaction Routing to Com-plete is activated. CICS only uses its internal session characteristics (that may differ from those defined in the logmode table) instead of the VTAM mode definitions but they are required by Com-plete.
5. Due to length differences between CICS transaction codes (4 bytes) and Com-plete program names (8 bytes), it is necessary to create a cross-reference table in Com-plete (see Com-plete Considerations) to match the transaction name trnm as known in CICS with the true Com-plete program name.
6. The CICS CRTE transaction is also supported thus allowing CICS users to invoke Com-plete applications that do not have a TRANSACTION definition. However, corresponding URTETB entries must be coded also for these transaction names.

Generic Resource names

Any VTAM application program running on a VTAM that is connected to the MVS coupling facility can use a generic resource name. VTAM keeps a map of the application programs that are members of each generic resource name. VTAM distributes incoming sessions that are initiated using a generic resource name among all members mapped to that name. A generic resource name may be specified by means of the VTAMGENERIC sysparm.

The following restrictions apply to generic resource names:

- An application program can use one generic resource name at a time.
- Generic resource members using the same generic resource name must have the same networkidentifier (NETID).
- Generic resource names must be unique within a single network. If your network has multiple sysplexes, generic resource names must be unique throughout all sysplexes.
- A USERVAR and a generic resource name cannot have the same name.
- An ALIAS application program and a generic resource name cannot have the same name.
- A name that is being used as an application program network name cannot be used as a generic resource name.

LOGMODES

Session parameters for the different device types or LU6.2 sessions are read directly from VTAM or its libraries, eliminating redundant specifications and possible inconsistencies between specifications. This requires correct and entire specification of the parameters in the logmode entries, since these override the specifications in TIBTAB.

APPC Interface

The APPC Interface is described under the following headings:

- Concepts
 - Implementation
 - Requirements
-

Concepts

APPC is a communication protocol specifically designed for distributed processing. While 3270-based communications are hierarchical (controlled by the host), APPC is peer-to-peer, meaning that programs communicate as equals. APPC-based applications send and receive data directly, without using 3270 screens. In APPC, the communication between two programs is called a conversation. The program that initiates the conversation is called the client, and the program that responds to the client is called the server. A distributed application is an application that requires programming from different places. A transaction program (TP) is one of the programs in a distributed application. A transaction is a business deal cooperatively completed by two or more transaction programs.

An APPC application can be designed to provide the functions of a 3270-based application, but without the same performance and reliability problems associated with 3270 communications. APPC programs can communicate with each other no matter which API is used by either side.

APPC data transfers are not restricted by screen size. Applications can send up to 32 kbytes of data in a single send. This alone greatly increases the performance of APPC over 3270. Additionally, APPC automatically buffers the data that is sent to the partner program. Buffering is performed to optimize network data flow for client/server applications. If a number of records are sent using several send calls, the data can be sent as a single network flow.

Because APPC is a peer-to-peer protocol, both APPC applications must agree on which communication flows will be used for the conversation. The communications flow includes who will start the conversation, what each side will say, when and how they will say it, and who will end the conversation. With APPC, any supported platform can have applications that function as either the client or the server.

APPC does not assume that applications will provide data in a specific format, since APPC can transport data in any format. When communicating between platforms, APPC does not perform data conversions. Although data conversion routines are simple to code, the fact that conversion may be needed is often overlooked when first developing APPC applications. The developer needs to include the ability to convert character, integer and floating point data when sending data between platforms.

Similarly, binary data must be handled differently in APPC. 3270 applications do not have to deal with binary data, since all data is expanded to text format for the 3270 screens. When migrating a 3270 application to APPC, it is often necessary to expand the binary data to text before sending it to the partner platform. Since different platforms store binary data in different formats, sending binary data in its native format across platforms can cause application errors.

There are several ways to ensure that both partners can correctly handle transported data:

- An organization can define a standard for all data that is exchanged, so an application can convert between the standard and the format for the platform on which the application is running.
- All numbers can be sent in character form rather than binary.
- Use system and language tools that help translate data to be sent to other machines.

Implementation

Com-pletes APPC Interface allows online programs to act as servers in an APPC conversation. In general the server program running under Com-plete does not know or does not need to know whether it is communicating with a terminal or an APPC partner. The data transformation is done later in the corresponding device modules according to TIB specifications. Existing 3270 application programs might work correctly as APPC server TPs. Clients must of course be to understand the existing logic of data flow. No 3270 control data is sent unless the TP inserts it documentationally (Write Special).

Requirements

To enable the Com-plete APPC Interface , the APPC=YES parameter must be present on the APPL definition of Com-plete and LOGMODE entries for SNASVCMG and a user LOGMODE must have been included in an active logmode table.

A buffer pool with an element size of 32K, location ANY must be specified in SYSPARMS for the APPC Receive Buffers.

The TP name length for servers running under Com-plete is restricted to 8 characters. TP names longer than 8 characters are truncated to 8.

ULOG is invoked prior to attaching the requested TP. A valid userID/PASSWORD combination must be present in the security fields of the ATTACH request. If logon fails the ATTACH is rejected with the corresponding sense code. On normal or abnormal termination of the requested TP the standard logoff procedures are invoked before the conversation is deallocated.

Com-plete fully supports BASIC and MAPPED conversations with Synclevel=None. Synclevel=CONFIRM is currently supported by the interface but API support is not yet available. CONFIRMD requests are generated automatically by the interface when appropriate.

Application Data GDS(12FF) is expected after the ATTACH FMH-5. If a User Control Data GDS (12F2) is found Com-plete assumes this is a CICS Transaction Routing Request and activates the supplied transaction URTE. UserID/PASSWORD in the security fields are then ignored. They will be taken from CICS input data.

CICS / Com-plete Transaction Routing

Com-plete may act as a CICS Application Owning Region (AOR) for CICS Transaction Routing. This allows terminals defined and connected to a CICS system to run with applications defined and executed in Com-plete. CICS users may acquire an ISC APPC-Type link to Com-plete and execute Com-plete applications from a CICS terminal. Com-plete behaves toward the CICS system as another CICS. The implementation of Transaction routing to Com-plete on a CICS system is no different from implementing CICS-to-CICS Transaction Routing. CICS versions 2.1 to 4.1 are supported.

This chapter covers the following topics:

- CICS Considerations
- Transaction Parameters
- Logon Security
- Programming Notes
- TRACES

CICS Considerations

CONNECTION, SESSION, TRANSACTION and PROFILE definitions must be coded in the CICS system. Recommended specifications :

```

CONNECTION
.....
Netname           : ==> Com-plete ACBNAME
.....
ACcessmethod     : ==> VTAM
PRotocol         : ==> APPC
SInglesess       : ==> No
DAstream         : ==> User
RECORDformat     : ==> U
.....
SESSION
.....
MOdename         : ==> modename      (1)
PRotocol         : ==> APPC
.....
TRANSACTION      (3)
.....
REMOtEName       : ==> trnm          (2)

```

Notes:

1. Mode modename must be defined in a mode table before Transaction Routing to Com-plete is activated. CICS only uses its internal session characteristics (that may differ from those defined in the logmode table) instead of the VTAM mode definitions but they are required by Com-plete.
2. Due to length differences between CICS transaction codes (4 bytes) and Com-plete program names (8 bytes), it is necessary to create a cross-reference table in Com-plete (see Com-plete Considerations) to match the transaction name trnm as known in CICS with the true Com-plete

program name.

- The CICS CRTE transaction is also supported thus allowing CICS users to invoke Com-plete applications that do not have a TRANSACTION definition. However, corresponding URTETB entries must be coded also for these transaction names.

Terminals to be used in Transaction Routing may be autoinstall-type terminals or TERMINAL definitions. Com-plete always requests CICS to ship the terminal definitions so all terminals must be defined as shippable:

```
TYPETERM
      SHIppable      : ==>Yes
.....
```

Com-plete Considerations

Before Transaction Routing can be activated, a conversion table (module URTETB) must be assembled and link-edited in the Com-plete load library. URTETB is loaded at Transaction Routing initialization and referenced for each new transaction. The Com-plete Transaction Routing Program (URTE) searches this table for the CICS transaction name and gets the Com-plete application program name to be started. If the CICS remote transaction name is not found in URTETB, or URTETB has not been loaded Com-plete returns an error to the CICS user and drops the session.

The basic format of URTETB is 1 CMROUTE TYPE=START macro call followed by any number of CMROUTE TYPE=ENTRY macro calls that specify the CICS transaction name (*trnm*) and the corresponding Com-plete program name (*pgmname*).

```
URTETB  CSECT
        CMROUTE  TYPE=START
        CMROUTE  TYPE=ENTRY ,CICSTRAN=trnm ,COMPROG=pgmname
        .
        .
        CMROUTE  TYPE=END
        END
```

Notes:

- CMROUTE macro and a sample URTETB module are included in the distribution SOURCE data set.
- trnm* can be equal to *pgmname*.
- pgmname* can further specify program parameters after the program name:

```
CMROUTE TYPE=ENTRY ,CICSTRAN=UQQ ,COMPROG='UQ Q ,JB=COM'
```

Transaction Parameters

When a CICS transaction is routed to Com-plete, the Com-plete program, as determined by URTETB, can also have parameters passed to it. There are 2 ways of specifying these parameters:

Static (hardcoded) in the COMPROG parameter of the CMROUTE macro (see note 3 above)

Dynamically entered in CICS after the remote transaction name in the standard CICS manner. These override eventual hardcoded parameters specified by method 1.

Logon Security

The CICS userID may be passed to Com-plete as basis for logon. If this user is invalid to Com-plete, ULOG asks for a valid userid/password before the requested transaction is started.

Programming Notes

Some Com-plete applications dependent on attention interrupt to terminate multiple writes (a loop of Write Returns to periodically refresh a screen) are not supported neither for native APPC nor for Transaction Routing. The partner transaction can only flag an interrupt when it gets the Send Token (sent by a Write Conversational), what never occurs. Examples are

```
UQ M,RR=n
```

```
USTOR LOOP
```

Com-plete can currently act only as SERVER in APPC sessions; it can only start local transactions attached from external clients. Programs running in Com-plete are not able to attach remote applications yet.

Indirect routing to Com-plete a routing path where other CICS systems are connected between the primary (CICS) and the secondary (Com-plete) systems) is not supported. Similarly, Com-plete cannot be used as an "indirect" system to enable routing between 2 CICS systems.

TRACES

To aid in problem determination the VTAM Interface provides various trace facilities:

1. snTIB Trace - retains the last 12 entries in the TIB. The interpreted entries can be seen in UUTIL MO TS together with the RPL entering PF11.
2. VTAM in-core trace - a wraparound table that contains basically the same entries as the TIB trace. Normally entries from all TIBs are stored, but it can be restricted for 1 specific TIB. This trace can be accessed only with USTOR. It resides after module TLAMTRAC, addressed by label ALAMTRAC in COMREG.
3. Extended Trace: This trace, available only for the APPC Interface and Transaction Routing Application is written to an external file defined by the SYSTRACE DD statement in the startup procedure. In general it points to SYSOUT. The file can be opened and closed online. Entries can be restricted to a specific TIB. The extended trace is an important part of the documentation when reporting problems of the APPC Interface to Software AG.

The VTAM In-core and Extended Traces are normally deactivated. To activate the In-core Trace use UUTIL TO (trace options) and type 'Y' for VTAM trace. To activate the Extended Trace, VTAM trace must also be set to 'Y' and the options 'OPEN EXTENDED TRACE' and 'PRINT BUFFERS AND DATA AREAS' also be set to 'Y'. For more details about traces refer to the UUTIL section in the Com-plete Utilities documentation.

LIBRARIAN (MVS Only)

The Com-plete LIBRARIAN interface, distributed as a feature of the Com-plete online editor utility in MVS only, allows Com-plete's online editor to access files stored within ADR's (Applied Data Research's) LIBRARIAN. However, before this interface can be used to access and modify LIBRARIAN members, the following installation procedures must be performed.

Step 1: Link edit and zap the LIBRARIAN module.

Two load modules containing the LIBRARIAN access routines must be placed on the Com-plete STEPLIB library. The FAIR (File Access Interface Routine) load module is used for retrievals only; COMLIB is used for updating LIBRARIAN members. Proceed as follows:

1. Copy the FAIR load module.

Using the IEBCOPY batch utility, copy the FAIR load module from the LIBRARIAN local library to the Com-plete STEPLIB library COM.USER.LOAD. (FAIR is provided by ADR as part of LIBRARIAN.) This routine is used as distributed by ADR by UEDIT to retrieve members from LIBRARIAN libraries.

2. Link edit COMLIB.

Use the following linkage editor example to link edit the COMLIB load module into COM.USER.LOAD. COMLIB is used by UEDIT to update a LIBRARIAN member when a SAVE command is entered to UEDIT.

```
//COMLIB EXEC PGM=IEWL,PARM='XREF,LIST,LET,NCAL'
//SYSLIB DD DSN=librarian,DISP=SHR LIBRARIAN LOAD LIBRARY
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLMOD DD DSN=COM.USER.LOAD,DISP=SHR
//SYSLIN DD *
INCLUDE SYSLIB(librarian)
ENTRY BEGIN
NAME COMLIB(R)
/*
```

3. Zap the COMLIB load module.

Zap the COMLIB load module produced in pint 2 above to change the DD names from SYSIN and SYSPRINT to LIBIN and LIBPRINT, respectively. To do this, use the linkage editor output from point 1 above to find the displacement of the references to "CARD" and "PRINTER". The external reference location is the beginning of the DCB for SYSIN and SYSPRINT. Add decimal 40 to these displacements in order to locate the DD names. Then, using the following superzap control statements, change the first three characters of the DD names from SYS to LIB.

```
NAME COMLIB CS1500
VER XXXX E2E8E2 SYS SYSPRINT DD name
REP XXXX D3C9C2 LIB
VER YYYY E2E8E2 SYS SYSIN DD name
REP YYYY D3C9C2 LIB
```

where:

XXXX	is the displacement of the external reference to "PRINTER" plus decimal 40.
YYYY	is the displacement of the external reference to "CARD" plus decimal 40.

Step 2: Modify the Com-plete sysparms

The Com-plete sysparms must be modified, causing Com-plete to include FAIR and COMLIB as resident programs. Do this by adding the following statements to the SYSPARM member used to set Com-plete options:

```
RESIDENTPAGE=FAIR
RESIDENTPAGE=COMLIB
```

In addition, the LIBRARIAN sysparm must be specified. See the chapter Initialization - Com-plete Startup Procedure for more information on the sysparms used in the Com-plete startup procedure.



Warning:

Do not modify the COMLIB load module used by Com-plete while Com-plete is running. During normal use of the LIBRARIAN interface by UEDIT, these modules are refreshed in the Com-plete resident program area. Increasing the amount of storage required by these modules can have catastrophic results.

Step 3: Set the spooling and printing options.

All error and informational messages issued by LIBRARIAN while processing a member are spooled to the TID defined in the TIBTAB as the hardcopy device for the terminal using LIBRARIAN. If no hard copy device is defined, the messages are spooled to the user's terminal.

By using superzap, the spooling of messages can be changed. To do this, obtain a linkage editor output for UEBP (or if a link edit was not done, use the HMBLIST batch utility to obtain a map of the UEBP load module). Using this map, find the entry point named "OPTION" in the CSECT U2EDCLIB. Using this displacement, use the following superzap control statements to set the desired spooling options.

```
NAME UEBP U2EDCLIB
VER XXXX 88   Default spool settings
REP XXXX NN   Desired spool settings
```

where XXXX is the displacement within the U2EDCLIB CSECT to the entry named OPTION.

XXXX	is the displacement within the U2EDCLIB CSECT to the entry named OPTION.
------	--

Option	Value	Function
x'80'	1...	Spool all messages.
x'40'	.1..	Spool only error messages.
x'08' 1..	Spool to hard copy TID.
x'04'1..	Spool to user's TID.

Step 4: Modify the Com-plete job stream JCL.

The two LIBRARIAN load modules made resident within Com-plete require additional storage from within the Com-plete region. You must therefore increase the REGION parameter for the Com-plete job step. In the case of VS1, the size of the partition in which Com-plete runs must be increased. The increase in size should be the combined sizes of the two load modules.

The Com-plete execution procedure must be modified to include DD statements for the files required by LIBRARIAN and also for each LIBRARIAN library to be accessed by UEDIT.

LIBRARIAN requires two files. The first is a sequential file (LIBIN) used to pass LIBRARIAN control statements and modified source statements to the LIBRARIAN interface module (COMLIB). This must be defined as an 80-byte fixed block file and must be large enough to contain the maximum number of control statements that will be passed to COMLIB.

The second sequential file (LIBPRINT) is used by COMLIB to pass informational and error messages back to UEDIT. This file must be defined as a 133-byte fixed block file with the first character of each record used for ANSI carriage control.

Either of the required files can be temporary files. The LIBPRINT file can be a dummy if no messages are desired. Note that LIBPRINT can not be defined as a SYSOUT file.

Examples of LIBIN and LIBPRINT DD statements are:

```
//LIBIN      DD UNIT=SYSDA,SPACE=(CYL,5),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//LIBPRINT DD UNIT=SYSDA,SPACE=(CYL,5),
//           DCB=(RECFM=FB,LRECL=133,BLKSIZE=2660)
```

or:

```
//LIBPRINT DD DUMMY
```

Each LIBRARIAN library to be used by UEDIT must be defined in the Com-plete job stream JCL and identified in the UEDIT table of library IDs and corresponding file names (UEDTB1). For LIBRARIAN files in the UEDTB1 table, the file name field specifies the corresponding DD name in the Com-plete job stream JCL, not the file name.

An example set of UEDTB1 entries and JCL DD statements follows:

In the Com-plete JCL, add the DD statement:

```
//MYLIBLIB DD DSN=USER.LIB.MYLIB,DISP=SHR
```

the corresponding UEDTB1 entry would be:

```
CMEDTB1 ID=ML,DSN=MYLIBLIB,ACM=LIBRARIAN
```

Step 5: Name the user exit (optional)

An optional user exit is available that allows you to enforce installation standards for the LIBRARIAN -SEL and -ADD control statements. This user exit must be named UXEEEX4. Control is passed when the SAVE operation is started to inspect the -SEL or -ADD control statement.

See Security and User Exit Facilities for more information about the coding and installation of UXEEEX4. In addition, an example of UXEEEX4 is provided in the COM.SOURCE distributed library.

VSE LIBR Service

The interface between Com-plete and LIBRARIAN is performed by the module TLSRLIBR.

TLSRLIBR, which handles the LIBRARIAN interface for both VSAM and non-VSAM libraries, was implemented using IBM-documented macros to access and update requests.

Since the LIBRARIAN interface uses the FILE-NAME as a means to relate a library to a DLBL/EXTENT, it is necessary to build and keep a FILE TABLE to maintain the DLBL/EXTENT information about the libraries to be accessed when using the Com-plete utilities. The FILE TABLE is built with information from the LABEL AREA and information from UEDTB1 initially; it is then expanded when a Com-plete utility attempts to access a library that does not have information in the LABEL AREA or UEDTB1.

The program U2SPIT, which must be specified in the sysparms (STARTUPPGM=U2SPIT), allocates storage for the FILE TABLE based on the number specified in the sysparm MAXLIBS, and anchors the address of the table in a field within COMREG.

Note:

The specification of STARTUPPGM=U2SPIT is required for VSE.

U2SPIT will call U2SPLA to build the FILE TABLE at initialization of Com-plete. U2SPLA does the following:

- Processes the STANDARD LABEL AREA, searching for labels defined as SD (in the DLBL). Note that tape labels and VSAM labels are ignored.
- Determines if the selected file is a library using the LIBRARIAN macros.
- Builds one entry in the FILE TABLE for each library defined in the STANDARD LABEL AREA. This entry maintains the following information: FILE-ID, FILE-NAME, and VOLUME SERIAL NUMBER.
- Repeats the above process for the PARTITION STANDARD LABEL AREA and the PARTITION TEMPORARY LABEL AREA.
- Processes UEDTB1 and builds new entries for files that were not in any of the LABEL AREAs.

The program U2SPLA is called during Com-plete execution by utilities requiring that a FILE-NAME be related to both a FILE-ID and VOLUME SERIAL, which are specified by the user (via utilities such as USERV, UEDIT, and UED). If the FILE-ID is not in the FILE TABLE, U2SPLA will add a LABEL to the PARTITION TEMPORARY LABEL AREA and build a new entry in the FILE TABLE (after verifying that the FILE-ID specified belongs to a valid library).

Note:

In order to access VSAM libraries, it is necessary to have them defined by either UEDTB1 or function UL of UUTIL.

Using VSAM with Com-plete

This chapter covers the following topics:

- Introduction
 - VSAM Record Sharing and Integrity Options
 - Using VSAM Files Online
-

Introduction

VSAM is an IBM access method that provides three file types, secondary indexing capabilities, and a useful utility referred to as Access Method Services (better known as IDCAMS).

The three file types, or clusters in VSAM terms, are:

- KSDS Key Sequenced Data Sets;
- ESDS Entry Sequenced Data Sets;
- RRDS Relative Record Data Sets.

The Key Sequential data sets are indexed data sets whose key is some variable or combination of variables in a file. The file is composed of two parts (actually two subfiles):

- Index component;
- Data component.

The index component contains the keys and a pointer to the associated data record. This pointer is often referred to as an RBA or Relative Byte Address. The data component contains the data record associated with the key. Note that the primary key must be unique within the file; however, any secondary keys (or alternate indices in VSAM terms) may have duplicates. KSDSs can be addressed either sequentially, randomly, or skip sequentially.

The ESDSs are sequential files; they may be addressed randomly through a Relative Byte Address (RBA) from the beginning of the file.

The key for a RRDS is the Relative Record Number, much like an Adabas ISN. In other words, if you want the third record in the file, the key would be three.

All VSAM files are divided up into logical units referred to as control intervals and control areas. A control interval consists of what could be considered a blocked record, i.e., one with multiple records per control interval. A control area is nothing more than a block of control intervals. Records may be variable or fixed lengths and may span control intervals.

VSAM has a much more efficient record storage concept for its KSDS than for other indexed access methods. It automatically builds multiple levels for its index components, and also handles new insertions, which cause other records to be shifted in an efficient manner.

The buffering services and sharing of control blocks are of great use in a multi-region/environment.

One of the capabilities of VSAM is its record sharing capabilities, which must be taken into consideration when using VSAM in an online environment. These capabilities and restrictions are discussed below.

From the application standpoint, the use of VSAM with Com-plete is a very easy task. The application programmer writes standard OS/VS COBOL VSAM I/O statements to access any VSAM files - just like writing batch programs using COBOL. The only special processing the application programmer must be concerned with is the checking of VSAM status codes and the writing of Com-plete screen I/O calls.

Important:

The file status codes must be checked after every VSAM I/O. For instance, if you do not check the VSAM status code after an OPEN statement and a bad open was encountered, you will get a "0C4" completion code when you try to reference the I/O area. This completion code indicates that addressability to the I/O area is not established until a valid open has been accomplished.

VSAM can provide data integrity at the control interval level (that is, VSAM reserves a control interval for one user and does not let any other users access this control interval until the first user releases this control interval). The types of integrity that VSAM provides are based on the share options specified at the time the clusters are defined. If a record is currently held by a user, a "93" status code is returned and the I/O call should be reissued. VSAM does not queue I/O calls that cannot be serviced. In other words, if a record is held by another task, VSAM will tell you about it, but it will not wait for that record to be freed before returning to you.

Normally, the procedure would be to issue a Com-plete rollout function (i.e., wait) and then reissue the call. If after several tries (a maximum of 10 times) the record is still not free, return a message to the screen indicating that the record is currently held and an attempt should be made later.

As an alternative, you can request Com-plete to serialize I/O requests for a file, thus avoiding a record not being available due to its being held by another user of the same Com-plete. For details, please refer to the description of the UUTIL online utility in the Com-plete Utilities documentation. The status codes and reasons that are most often returned are listed in the following table:

Status Code	Reason	Action
90	Bad OPEN: File was not closed properly by a previous user.	Run an IDCAMS VERIFY
93	Record or resource is not available Either another user has the record or the file is being used by another task in another region. On an OPEN, there is not enough virtual storage in Com-plete's region.	Ensure no other job is using the file. Use the FM function of UUTIL to check if the file is open online. Ascertain if it may be useful to request I/O serialization by Com-plete.
95	Logic error in I/O call. Usually your FD in COBOL does not match that defined in the VSAM cluster definition.	
96	The file is not defined correctly in Com-plete.	Use the FM function of UUTIL to define the file.

The *COBOL Programmers's Guide* provides a more detailed description of VSAM status codes and should be consulted. In addition, be aware that COBOL status codes have a different meaning based on the request type, that is, Open, Read, Write, etc. The *COBOL Programmer's Guide* also contains a table indicating the associated VSAM error codes and includes definitions of the codes.

There are several considerations to be made in cataloging a VSAM "DDN" to Com-plete. They include:

- Locally shared vs. non-shared resources;
- COBOL use of VSAM string processing;
- Defining buffer areas of VSAM files;
- Defining share options in the VSAM cluster definition.

VSAM Record Sharing and Integrity Options

Overview

The VSAM record sharing options are defined in the SHARE OPTIONS parameter when defining a cluster. Although VSAM allows sharing options for both across-region and across-system, only across-region options are discussed below. Com-plete users in different threads fall into the across-region category.

The share option "1,3" provides Com-plete read/write data integrity, that is, multiple users can access a data set for read processing while another user has it for update processing. Under this option, a "read" user cannot process a control interval that is currently being updated (read integrity). No two users can access the same control interval for update, thus providing write integrity; however, they can simultaneously update data in different control intervals.

The share option "2,3" is very similar to the "1,3" option, but it does not provide read integrity. In other words, it is possible for a read user to access the same control interval that another user is updating, thus losing read integrity. There is no problem in updating with this option because VSAM ensures that only one user has a control interval for update.

Note:

If you specify options "2,3" but access the data in a shared resources environment (that is, the Com-plete DDN is cataloged with LSR instead of NSR), VSAM forces share options "1,3". So if you want a "2,3" option, you must catalog the DDN with NSR, non-shared resources.

You are recommended to use VSAM option "2,3" if you want to read a file in a batch program while the file is still online.

There are two other options: "3,3" and "4,3". Neither one of these provide read or write integrity. It is up to you to provide this integrity (which requires Assembler level coding). You are not recommended to use these options unless it is absolutely necessary.

Note also that a control interval is not released from an update held status until the update user either accesses another control interval, issues a ENDREQ assembler macro, or closes the file. This can lead to problems where a record is read for update and sent to the screen. The control interval is tied up until the user finally responds, unless the program closed the file before returning to the screen. There are two options:

- Opening and closing a file on every scheduling of an update (never do this on a read-only program);
- Allowing the records in a control interval to be tied up until the user responds.

If you choose to use the second approach, inform the user that the records were held and that an attempt should be made later.

Buffers

Another important factor in the performance of VSAM processing is the use of buffers. VSAM automatically allocates one index buffer and two data component buffers for each string of a KSDS. (A string is a unique access path to a portion of a file.) Each string requires that VSAM maintain the position in the file. The amount of space required for a buffer is determined by the control interval size. So the larger the control interval size, the larger the buffer space. This also means that you have more records available. If the user is using sequential processing, then performance can be increased by using more data component buffers. If a user is using direct processing, then performance can be increased by using more index buffers. More index buffers are required when direct processing is used because VSAM accesses the index component for every request and refreshes the data component on every request. Thus, the more index control intervals kept in memory, the fewer I/Os needed to access records. Under sequential processing, VSAM does not access the whole index on every request but will instead use the horizontal pointers in the sequence set (the lowest level of an index) to access the next record. VSAM can use additional data buffers to do read-ahead functions and thus have data available for future requests. Note that sequential processing does not refresh its data component buffers on every request.

A complete discussion on optimizing performance and VSAM buffering techniques is available in the *VSAM Programmer's Guide* and should be consulted. This information is extremely important when you define the size and number of buffers to allocate when using the LSR (Locally Shared Resources) option. One difficulty encountered in using COBOL, Com-plete, and VSAM is the consistency and meaning of terminologies used in the different documentations involved. The following information should aid you in resolving this problem.

- COBOL does not support skip sequential processing, so there is no need to specify the SKP option when cataloging a DDN to Com-plete.
- The way COBOL determines whether you are going to update, or simply read a file is based on how a file is opened, that is, I/O=update, INPUT=read-only, OUTPUT=update.
- COBOL does not support addressed processing, but does support key processing.
- COBOL does not support multiple string processing. However, multiple strings to a data set are acceptable because of the concurrent running of programs accessing the same file from different threads.
- There is one RPL (Request Parameter List) for sequential processing and a second RPL for direct processing.
- The definitions of the option codes you specify when cataloging a VSAM DDN to Com-plete are explained in the *VSAM Programmer's Guide*.
- You must consult the following documentations when using VSAM, COBOL, and Com-plete:

- Access Method Services
 - VSAM Programmer's Guide
 - VSAM Programmer's Guide for Advanced Applications
 - OS/VS COBOL Compiler and Library Programmer's Guide
 - OS/VS COBOL documentation
 - Com-plete Application Programming documentation
 - Com-plete System Programming documentation
 - Com-plete Utilities documentation (particularly ULIB)
- The following table relates some COBOL terms to VSAM terms.

COBOL Term	Associated COBOL Verbs/Clauses	VSAM Term
Random processing	ACCESS IS RANDOM or ACCESS IS DYNAMIC, READ, WRITE, REWRITE, DELETE	Direct processing
Sequential processing	ACCESS IS SEQUENTIAL or ACCESS IS DYNAMIC, READ NEXT, START, WRITE, REWRITE, DELETE	Sequential processing
File status code	FILE STATUS IS (COBOL interprets the VSAM codes and returns its own codes)	Return Code.
Current record pointer	N/A	Position within file
READ	READ	GET macro
WRITE	WRITE	PUT macro
REWRITE	REWRITE	PUT macro with update
DELETE	DELETE	ERASE macro

Using VSAM Files Online

The following sections contains notes on using VSAM files online.

- VSAM Files
- Alternate Indices

VSAM Files

1. If the file is to be updated, use MACR = (..NDF..). This slows down performance but ensures data integrity.
2. Make any file known to Com-plete reusable unless you are going to define an alternate for it. But remember that whenever that file is opened for output, the file is automatically emptied.
3. Make the data CI size small (1024,2048,4096), index CI = 512.
4. Avoid CA and CI splits by using FREESPACE for a volatile file.
5. Use spanned records with variable length records where only a few records are very large.
6. Avoid files where the records are deleted from the bottom and added to the top.
7. Do not specify ERASE unless the data is sensitive and all traces of it must be zeroed out.
8. STATUS-KEY = 96 (COBOL) and ONCODE = 1027 (PL/1) mean that the record specified is being held by another program that has the file opened as I/O.
9. ONCODE = 1028 (PL/1) means that after the file was opened by your program, someone else put the file in batch status and your program then tried to read the file.
10. Do not use IDCAMS to verify files owned by Com-plete while Com-plete is running. Set them to BTCH status first.
11. Use SHAREOPTIONS (2,3) when defining a cluster in order to avoid problems with one program trying to access a file as I/O while another program has that file accessed as input.
12. The application programmer is responsible for VSAM file backup and recovery. The best method to backup a file is to use a tape. The second best is to REPRO the file to another disk pack. It is not a good idea to back up a VSAM file to the same pack since a head crash or other disk pack error could wipe out both the real file and the backup file. It is also better to use REPRO instead of IMPORT/EXPORT to back up a file, because REPRO can reorganize the file at the same time (to reorganize the file, you must REPRO it back).
13. Use the appropriate VSAM prefix for that pack when naming a VSAM file (VSAM2. for MVS002). The second level name should be the valid prefix for that department. Should you need to change the name of any VSAM file, use the ALTER command of IDCAMS. Name the data and index components as well as the cluster.
14. Do not issue terminal I/O while holding any VSAM resources, for example, between GET for update and UPDATE, or in the middle of sequential VSAM processing.
15. Name both the data and index component as well as the cluster. This makes it much easier if you have to perform an ALTER on the VSAM file.
16. Thread-lock all programs using the same file(s) to the same thread, or request I/O serialization by Com-plete.

17. If you get an ONCODE = 92 when you open the VSAM file and the accompanying message says "DATA SET UNAVAILABLE", the file is probably offline (batch status). Another reason may be that the DDN is not defined to Com-plete.
18. If you get an ONCODE = 92 and the accompanying message says "DATA SET NOT PROPERLY CLOSED", it means that a job went down in the middle of VSAM processing and left the file open. To avoid this, include an IDCAMS verify step in the Com-plete initialization procedure, specifying all VSAM data sets that may be affected. Com-plete closes all VSAM files in use during termination processing, irrespective of whether it is terminating normally or due to an ABEND or CANCEL.

Alternate Indices

1. Use SHAREOPTIONS (2,3) for both the base cluster and the alternate index.
2. When using a generic key, you may want to add a high value record to the file to avoid end-of-file problems (particularly if you get an ONCODE = 1026).
3. A base cluster cannot be reusable.
4. The base cluster's DDN must be five characters long and each alternate index built upon that base cluster must use those same five characters plus a single digit.
5. In both the base cluster's DDN definition and the alternate index's DDN definition, use MACR=(..NSR..).
6. Make sure you define the path to Com-plete, not the alternate index. If you read the file using the path, the data is put in the base cluster. If you read the file using the alternate index, however, all the keys of the base cluster contain that alternate index key.
7. The recordsize (RECSZ) of the alternate index cluster is determined by the number of non-unique keys expected. The data consists of five bytes used by VSAM, the alternate key, and all of the base cluster keys that have this alternate key. Watch for any one alternate key having a large number of non-unique occurrences (the alternate KEY = blanks or zeros, etc.); this determines the maximum value of the record size.
8. In building the alternate index, IDCAMS performs an internal sort. If you have a large number of records in the base cluster, you must give IDCAMS a larger region and some additional JCL in order to build the indexes.

```
//BUILDALT JOB (990030,1,1,0),TIM,CLASS=C
//BUILD EXEC PGM=IDCAMS,REGION=600K
//SYSPRINT DD SYSOUT=A
//STEP1CAT DD DISP=SHR,DSN=CATALOG.VIPORES
//IDCUT1 DD DISP=OLD,AMP='AMORG',VOL=SER=IPORES,UNIT=3375
//IDCUT2 DD DISP=OLD,AMP='AMORG',VOL=SER=IPORES,UNIT=3375
//SYSIN DD *
BLDINDEX
        INDDATASET(VSAMIPOR.SYS1.TESTINDEX)
        OUTDDATASET(VSAMIPOR.SYS1.TESTALT)
        CATALOG(CATALOG.VIPORES/PWUPDATE)
```

9. Whenever you delete the base cluster, IDCAMS automatically deletes any and all paths to that base cluster.

10. When using the alternate index to look at the data, changing the alternate key by means of the base cluster, and then trying to look at the old alternate index key again, watch for this potential problem: VSAM looks at the data in the buffers (in this case the alternate index) first and use it without actually doing another read of the file. Your program then tries to read the file using the old alternate key and fails because that record is no longer there. The solution is to flush the buffers. One way is to have a high value record in your file and read it by means of the alternate key which will change the buffers of the alternate index. (Note that the above problem will give you an ONCODE = 1030.)
11. Freespace can also be defined for an alternate index and should be used if any records are to be added to the base cluster. CI and CA splits seem to cause problems for alternate indices.
12. When the alternate index is built (by IDCAMS), the records are in base cluster key order. However, any adds to the file after the base cluster is built are added to the end of the alternate index and are not in base cluster key order.
13. In PI/1, there is a built-in function called "SAMEKEY", which can be used if a VSAM file is being accessed by an alternate index that has non-unique keys. If any further records exist with the same key, it returns a "1"B.
14. Under PI/1, it is possible to define one alternate index path as forward and another path as backward. This would allow the user to not only page forward from any given key but also to page backward. One constraint is that a file declared as backward can not also be declared as GENKEY. You are therefore required to use the full key for the starting point of the backward read.
15. If you try to read past the end of the file using the alternate index, an ONCODE = 1030 is issued.

GDDM

This chapter describes how to install the Com-plete/GDDM interface that will enable Com-plete to use color graphics capabilities. Other documentation relevant to the installation of this interface is the *GDDM Release 4 Installation and System Management* (Publication Number SC33-0152)*, hereafter referred to as the *GDDM Installation Guide*. For the installation described in this section, you should be familiar with the GDDM documentation and refer to this section for clarification of the differences in the way GDDM interfaces to Com-plete and CICS. Com-plete and CICS are similar in that they are both teleprocessing monitors. This is the reason that Com-plete uses the same interface to GDDM as CICS. For this installation, then, generally follow the instructions concerning the installation of GDDM under CICS as described in the *GDDM Installation Guide*. This section supplies specific information for the Com-plete environment.

* Current name and number of documentation - these are subject to change.

Having finished the installation process, your installation will have the ability to run the various GDDM editors supplied with the various components of GDDM that you have licensed.

Note:

You need GGDM to install Natural GRAPHICS under Com-plete. See Adding DFHEAI and DFHEAI0 below.

This chapter covers the following topics:

- Installing GDDM
 - Com-plete Components for the GDDM Interface
 - Adding DFHEAI and DFHEAI0
 - VSAM, ADMF, DFHTSD
 - Altering the CICS environment
 - TIBTAB
 - Com-plete JCL Modifications
 - Com-plete Startup Parameters
 - UUTIL/ULIB Activities
 - User Calls
 - Execution
 - Performance Considerations
-

Installing GDDM

The first step is to install GDDM/PGF according to the installation procedures in the *GDDM Installation Guide*. This will result in the creation of four libraries (MVS):

- GDDMLIB non-executable load modules;
- GDDMSAM sample applications, macros, JCL;
- GDDMSYM sample symbol sets;
- GDDMLOAD executable load modules.

or it will create an equivalent set of VSE libraries if VSE is the operating system.

The reentrant code supplied for GDDM/PGF is approximately two megabytes. You must decide where this is to reside, and allocate the space required for it. To help you determine the best place for GDDM, refer to the *GDDM Installation Guide*.

The main consideration when deciding where to place the GDDM code is whether or not there is another subsystem (TSO, CICS, etc.) in your installation that will be using GDDM. If this is the case, you should consider putting GDDM in LPA (MVS) or other common system storage. If there is no other subsystem, it is probably better to place GDDM in Com-plete'S region or partition.

Once you have determined where GDDM is to reside, install it according to the procedures in the *GDDM Installation Guide*. In the *GDDM Installation Guide*, you will find a section on installing GDDM with CICS for MVS and information on how to install GDDM with CICS for VSE. Particular attention should be paid to the sections concerning the adding of the CICS command level stubs DFHEAI and DFHEAI0 (see Adding DFHEAI and DFHEAI0 below).

Com-plete Components for the GDDM Interface

The following table describes the Com-plete components of the GDDM interface, and how they are defined within Com-plete. During standard installation all DD names are with default values; however, these may have to be reviewed based on the usage of GDDM and for performance reasons. A sample list of programs which must be residentpaged is contained in member GDDMSAMP on the distributed source data set. This must be reviewed, particularly in relation to new GDDM releases, as new modules may have been included which can also be residentpaged.

Component	Description	Definition
ADMC	The Com-plete application program to invoke the GDDM Interactive Chart Utility.	ULIB CAT
ADMI	The Com-plete application program to invoke the GDDM Image Symbol Set Editor.	ULIB CAT
ADMV	The Com-plete application program to invoke the GDDM Vector Symbol Set Editor.	ULIB CAT
ADMP	The Com-plete application program to invoke the GDDM print utility.	ULIB CAT
TLCIINIT	Com-plete/GDDM initialization processing	RESIDENTPAGE
TLCIMAIN	Com-plete/GDDM control processing	RESIDENTPAGE
TLCISTG	Com-plete/GDDM storage processing	RESIDENTPAGE
TLCIPRGM	Com-plete/GDDM program processing	RESIDENTPAGE
TLCIICP	Com-plete/GDDM "interval control" processing	RESIDENTPAGE
TLCITERM	Com-plete/GDDM terminal I/O processing	RESIDENTPAGE
TLCIFILE	Com-plete/GDDM file I/O processing	RESIDENTPAGE
TLCITRND	Com-plete/GDDM "transient data" processing	RESIDENTPAGE
TLCITEMP	Com-plete/GDDM "temporay stg" processing	RESIDENTPAGE
TLCITRAC	Com-plete/GDDM trace processing	RESIDENTPAGE
TLCIFC	Com-plete/GDDM file processing	
DFHEAI	Com-plete/GDDM interface stub 1. Linked with various mods	
DFHEAI0	Com-plete/GDDM interface stub 2. Linked with various mods	
ADMI	GDDM Symbol Set File DD name	UUTIL FM
DFHTSD	Com-plete/GDDM "Temporary Storage" data set DD name	UUTIL FM

Adding DFHEAI and DFHEAI0

Com-plete supplies you with two modules called DFHEAI and DFHEAI0. These modules are direct replacements for the CICS modules of the same name. Therefore, the section in the *GDDM Installation documentation* entitled **Link-edit with DFHEAI and DFHEAI0** in **Tailoring GDDM step-by-step for CICS** must be repeated or initially done using the Com-plete replacement modules.

Take the sample job stream described in this section for linking GDDM with DFHEAI and DFHEAI0 and change the loadlib from which the modules are linked to point at the Com-plete load library. This will cause the Com-plete replacement modules to be included.

Be careful, however, that the various GDDM modules have not already been linked with the CICS modules, as a second include will simply make the module bigger and use the previously included modules. Example job streams are also provided for GDDM-PGF, GDDM-IMD and GDDM-IVU in the source library. Repeat the same procedure for each of these if this is applicable for your installation. If a sample GDDM job is not available, a sample job called GDDMLINK is supplied on the distributed source data set.

Please note that Natural GRAPHICS uses the system programmer's interface (module ADMASPC) to GDDM, which requires the modules DFHEAI and DFHEAI0. These modules must be taken from the Com-plete load library.

VSAM, ADMF, DFHTSD

Prepare your VSAM files as stated in the *GDDM Installation Guide*. You can use ADMF and other similar VSAM files under Com-plete, just as the function GDDM ESLIB describes.

You must create an additional VSAM file required by Com-plete's GDDM Interface. Its DDN to Com-plete must be DFHTSD. A sample MVS job stream to create it follows below:

```
//jobname JOB (JOB card information)
//JOB CAT DD DSN=usercat,DISP=SHR
//ALLOCEXEC PGM=IDCAMS
//dd1DD UNIT=SYSDA,VOL=SER=volser,DISP=SHR
//SYS PRINT DD SYSOUT=A
//SYS IN DD *
    DEFINE CLUSTER -
        (NAME(index.DFHTSD) FILE(dd1) -
        UNIQUE -
        VOLUMES(volser) -
        SHAREOPTIONS(3 3) -
        RECSZ(400 3000) -
        CONTROLINTERVALSIZE(4096) -
        KEYS(10 0)) -
    DATA -
        (RECORDS(300 100)) -
        CATALOG (usercat)
/*
```

After defining this file, it must be initialized. A sample job stream to initialize it would be:

```
//jobname JOB (JOB card information)
//VSAM EXEC PGM=IDCAMS
//SYS PRINT DD SYSOUT=A
//DD1DD DSN=index.DFHTSD,DISP=SHR
//STEP CAT DD DSN=usercat,DISP=SHR
//DATAD DD *
FIRSTONE THIS IS THE FIRST RECORD TO CREATE IT
/*
//DD3 DD UNIT=SYSDA,DISP=OLD,VOL=SER=VOLSER
//SYS IN DD *
    REPRO INFILE(DATAD) -
        OUTFILE(DD1)
/

// JOB GDDMTSDDEFINE DFHTSD
// DLBL UOUT,'FIRST.RECORD',0,SD
// EXTENT SYS005,xxxxxx,1,0,nnnnn,1<=====
// ASSGN SYS005,DISK,VOL=xxxxxx,SHR <=====
```

```

// ASSGN SYS004,SYSIPT
// EXEC OBJMAINT
./ CARD DLM=
./ COPY
FIRSTONE THIS IS THE FIRST RECORD TO CREATE IT
/*
// OPTION LOG
// DLBL IJSYSUC,'vsamusercatalog',,VSAM<=====
// DLBL DFHTSD,'GDDM.DFHTSD',,VSAM <=====
// ASSGN SYS007,DISK,VOL=xxxxxxx,SHR<=====
// DLBL INFILE,'FIRST.RECORD',0,SD
// EXTENT SYS007,xxxxxxx
// DLBL DFHTSD,'GDDM.DFHTSD',,VSAM
// EXEC IDCAMS,SIZE=AUTO
    DEFINE CLUSTER -
        (NAME(GDDM.DFHTSD) -
        VOLUMES(xxxxxx) - <=====
        SHAREOPTIONS(3 3) -
        RECSZ(400 3000) -
        CONTROLINTERVALSIZE(4096) -
        KEYS(10 0)) -
    DATA -
        (RECORDS(300 100)) -
        CATALOG (vsamusercatalog)<=====
    REPRO INFILE(INFILE ENV(BLKSZ(80) RECFM(FIXUNB) RECSZ(80))) -
        OUTFILE(DFHTSD)
/*
/&

```

All VSAM files must be defined to Com-plete using the FM function of online utility UUTIL. If you specify the files' data set names there, there is no need to include the DD statements in the Com-plete JCL.

Altering the CICS environment

The headings in the *GDDM Installation Guide* regarding the modification of CICS tables is of no interest to users of GDDM under Com-plete. CICS uses a series of tables to keep track of the files it uses and the destinations for certain terminal requests. Com-plete uses several different mechanisms for the same functions. For instance, Com-plete users catalog the VSAM files they use online with UUTIL FM. You can therefore ignore the sections in the GDDM documentation which describe the altering of CICS tables. The only CICS-like table Com-plete maintains is a file control table which is referenced internally by the Com-plete/GDDM interface. If you wish to define additional VSAM files for use with symbol sets, Com-plete will dynamically add entries into this internal table, thus freeing you from such table maintenance.

TIBTAB

The philosophy behind this is to reduce the amount of maintenance for the system programmer to an absolute minimum. Therefore, rather than having to define the attributes of different devices in different places (eg. LOGMODE and CICS TCT), Com-plete takes the LOGMODE for a device to indicate exactly what that device is.

Therefore, once all the device types are defined in the COMPLETE TIBTAB, when a terminal or its capabilities change, the only change necessary is an update of the LOGMODE for the device. Care must be taken with logmodes for devices to ensure that they indicate what the device can actually do. In this

case, the relevant indicator is an indicator that the device can support a Read-Partitioned-Query command. Once this is possible, the software can work out the rest. IBM supply various examples of LOGMODES for various device types.

In general, VDUs need not be specified in the TIBTAB as they can be allocated dynamically. Printers can also be allocated dynamically. However, it is sometimes better to define them in the TIBTAB to avoid having to reallocate them after a restart of Com-plete, unless a fully dynamic TIBTAB is specified (via TIBTAB=DYNnnnnn in the Com-plete sysparms). A sample TIBTAB follows. This defines two printers PRT1 and PRT2. A maximum of 100 terminals are available with this TIBTAB.

```
TIBTAB  TIBSTART NOTIBS=100
PRT1    TIB      1,VTAM,3287L,NAME=PRT1,OPT=(ACQUIRE,SHARE)
PRT2    TIB      2,VTAM,3287L,NAME=PRT2,OPT=(ACQUIRE,SHARE)
TIBEND
```

Note:

Terminals defined in the TIBTAB can be changed if information given in the LOGMODE conflicts with that specified in the TIBTAB.

Com-plete JCL Modifications

Com-plete must be able to find the various GDDM and Com-plete/GDDM interface modules. To make the GDDM modules available, the GDDM load library must be included in the Com-plete COMPLIB concatenation AFTER the library where the newly linked CICS/Com-plete-dependent modules reside (that is, the modules that are linked with DFHEAI and DFHEAI0).

To make the Com-plete/GDDM interface modules available, the Com-plete distributed load library must also be in the COMPLIB concatenation. However, you have the option of copying the relevant modules to a library which is already in the concatenation if the inclusion of the Com-plete distributed load library in the COMPLIB concatenation is not desirable.

Com-plete Startup Parameters

GDDM/PGF requires a considerable amount of work area. This will be allocated in the thread. Load as much of the code as possible outside of the thread to allow more room for the work area. The size of the thread to allocate will depend on the amount and type of work done. In a lot of cases this will be trial and error, so you are advised to start with the Software AG supplied default listed below. You must therefore review thread sizes to ensure that at least one is available that is big enough to run the various programs.

To reduce thread and catalog sizes, as much of the GDDM nucleus and code as possible should reside outside the thread. If GDDM is installed in the LPA, Com-plete can use these modules. However, if this is not the case, Com-plete can load them into its own nucleus via the RESIDENTPAGE sysparm. All eligible residentpage modules should be included using this parameter. A module is eligible when it is Reentrant and Reusable which, in fact, a large part of the GDDM nucleus is. The best thing to do is to look at the modules required to be defined in the CICS PPT. Take this list and delete any module not residentpage-eligible. The result is a list of modules which you must include in the Com-plete nucleus via the RESIDENTPAGE sysparm. A sample set of RESIDENTPAGE parameters is supplied in the GDDMSAMP member of the distributed source data set.

To enable the Com-plete VSAM interface to function, the VSAMDS sysparm must be specified. A starting value for this could be VSAMDS=2 indicating that a maximum of two VSAM files will be used for the duration of the Com-plete session. This is the current minimum that the Com-plete/GDDM interface requires to function. This can be increased as the need for more VSAM files arise. Please see the subsection **Performance Considerations** below concerning more VSAM parameters.

UUTIL/ULIB Activities

This subsection describes the various default parameters for the GDDM/Com-plete components for UUTIL FM and ULIB. If you have used the standard Com-plete installation procedure, these entries already exist. However, you may wish to change them for performance reasons.

Define the VSAM files to Com-plete using the FM subfunction of UUTIL. You must specify the following options:

For ADMF and DFHTSD:	Retrieval Update Add STRNO=2 MACRF=(DIR,NSR,NDF,SEQ)
----------------------	--

The following are the ULIB CAT parameters for the programs:

ADMI	IMAGE SYMBOL SET EDITOR	768K
ADMP	printer utility	768k
ADMV	vector symbol set editor	768k
ADMC	chart utility	768k
TLCIFC	VSAM file control table	(no size need be specified)

User Calls

The WRTSF function of Com-plete is available to users of graphics. See the Com-plete Application Programming documentation for details. It is strictly for use with extended function 3270's:

```
CALL WRTSF|D|C|R|      to perform a write structured field
                       to a specified TIB.
```

This CALL has the same structure as all the other writes. It is for performing write structured field commands. If you wish to perform a Read-Partition-Query, since this generates an input interrupt, use the WRTSFC option. You will get the results of this read in the input buffer. Be sure to use the READS form of the read CALL.

For loading symbol sets, use the WRTSFR option, as you do not need to wait for any input to occur. Alternatively, you can load programmed symbols using the UMAP utility LOAD PROGRAMMED SYMBOLS function (see the Com-plete Application Programming documentation).

The WRTSFD option is used when you want to set something and exit. For more information about the Write Structured Field command, consult the *IBM 3270 Information Display System documentation* (Publication Number GA23-0060) and the *3274 Control Unit Description and Programmer's Guide* (publication number GA23-0061).

When you use this form of the WRITE, Com-plete does NOT check for data to be output. The user program must ensure that the data being sent is valid for the WRF terminal command.

Execution

When executing GDDM/PGF in the Com-plete environment, use the CICS instructions in the GDDM and PGF documentations. Some differences exist, and these are discussed below.

CICS abends with alpha abend codes, that is, GDDM will ask for an abend G000. Com-plete converts that to abend 7000. If GDDM has issued an abend code that is alpha, Com-plete changes the zones to "F" and continues as though it was numeric.

Com-plete's interface abend codes are:

922	function not supported.
923	unrecoverable terminal write error.
924	no storage available in the thread
925	error in the ask time routine
926	error in ENQ or DEQ
927	error in loading or releasing programs.
928	error in initializing the graphics environment.

Performance Considerations

GDDM is resource-intensive, but some tuning can be done. Firstly, review the CPU TIME and REALTIME sysparms for Com-plete. These will avoid superfluous abends and messages if they are specified correctly.

GDDM makes quite heavy use of its VSAM files and therefore allocation of buffers to these data sets should be carefully reviewed with the expected usage in mind. VSAM itself handles buffer usage and therefore the only control you have is to allocate more or less buffers. This can be done on an individual file basis or the files can be grouped into a VSAM Local Share Resources pool (LSR). This is the most efficient in that all buffers are available for use for all files (given the restrictions of VSAM LSR, see the relevant VSAM documentation) as opposed to individual buffers which use storage even when the file is not in use. See the section **Using VSAM with Com-plete** and the Com-plete sysparms VSAMBUFFERS and VSAMRPL.

PANVALET (MVS only)

The Com-plete PANVALET interface allows a Com-plete terminal user to edit PANVALET library members up to Release 14 using the Com-plete online editor, UEDIT.

This chapter covers the following topics:

- Installation Overview
 - Installation Procedure (MVS)
-

Installation Overview

Before a PANVALET library can be accessed, two load modules must be created using the linkage editor. These modules are standard PANVALET routines provided by Pansophic Systems with additional modules added by the linkage editor. These modules provide compatibility with the Com-plete online environment and allow the Com-plete printout spooling facility to be used for spooling the PANVALET reports.

The PANVALET libraries to be accessed by UEDIT must be defined in the UEDTB1 table described in **Com-plete Files and Associated User Files**, and in **UEDTB1 Entry DSECT**.

Installation Procedure (MVS)

Step 1: Link edit and zap the PANVALET modules.

The member PANLNK contains an inline procedure to link edit the PANVALET modules. This procedure must be modified to use the load module library that contains the PANVALET modules at your installation. Do this by modifying the keyword parameter PANLIB default value in the PANLNK PROC.

Proceed as follows:

1. Link edit FG PAN23.

The PAN23 step is used to link edit FG PAN23, creating a load module on the Com-plete user load module library defined in the PANLNK PROC as LLIB. FG PAN23 is a module accessed by PANVALET during the processing of abnormal conditions. The execution of this link edit step prevents an abend of UEDIT (S806) from occurring because PANVALET could not find FG PAN23.

2. Link edit COMPAM.

The COMPAM step is used to link edit COMPAM, the load module used by UEDIT for read-access to the PANVALET libraries. COMPAM is identical to the PANVALET access method module PAM provided by Pansophic Systems.

3. Link edit COMPAN

The COMPAN step is used to link edit COMPAN, the load module used by UEDIT for write-access to the PANVALET libraries. COMPAN is identical to the PANVALET module PAN#1.

4. Link edit PVPVLAMS

The PVPVL step is used to link edit PVPVLAMS, PANVALET's general access module. PVPVLAMS is comprised of PVPVLAMS and Com-plete's PANVALET interface exit \$USTIMER. You must place this module in an APF-authorized library.

\$USTIMER functions as an interval timer routine that will do a ROLLOUT (if Com-plete), or a STIMER SVC (if TSO). The timer interval is currently set at two seconds.

The \$USTIMER module is supplied in source format. If this requires changes to the time interval, the module must be modified and reassembled.

5. Alter the print options

IMBLIST can be used to determine the offsets of the individual entry points in the module and the options modified as required. The zap has the format:

```

NAME COMPAN COMPAN
VER xxx yy    <-- old value
RER xxx zz    <-- new value

```

The entry point names and the functions of the five bytes are listed in the following table:

Entry Name	Function
OPTION	A switch used to control the printing of reports. Bit settings are as follows:
	X'80' Output all reports to the PANPRNT data set.
	X'40' Output no reports to the PANPRNT data set.
	X'20' Output only reports selected by the user code to the PANPRNT data set.
	X'08' Output all reports to the selected terminal.
	X'04' Output no reports to the selected terminal.
	X'02' Output only reports selected by the user code to the selected terminal.
	X'01' Select a user terminal for reports (otherwise, the terminal assigned as the screen-to-hardcopy terminal is selected).
SPPOS	Offset in the PANVALET user code to be tested for the X'02' option selected above.
SPCHAR	Value of the report selection code for the X'02' option selected above.
PRPOS	Offset in the PANVALET user code to be tested for the X'20' option selected above.
PRCHAR	Value of the report selection code for the X'20' option selected above.

Step 2: Modify the Com-plete job stream JCL

The Com-plete execution procedure must be modified to include the data definition statements for the files required by PANVALET, as well as for each PANVALET library to be accessed by UEDIT.

PANVALET requires that four data sets be identified by the JCL in addition to the libraries to be accessed by UEDIT. The first required data set is used to pass the PANVALET control statements to the routine COMPAN (PAN#1). This data set must be defined as an 80-byte fixed block sequential file, and must be large enough to hold the maximum number of control statements that will be required by PAN#1. This control statement file, identified by the name PANSEQ, serves the same function as the usual SYSIN DD statement used by PAN#1.

The other required statements are used for output of reports generated by PAN#1. These data sets must be defined as 133-byte fixed block data sets and identified by the names PANPRNT, PANDD2, and SYSPUNCH. The first character of each record is used for ANSI carriage control. Note that if these data sets are not defined as SYSOUT data sets, they should be defined with DISP=MOD and be large enough to contain all the reports generated by PANVALET.

All the required files can be temporary files. Examples of the PANSEQ and PANPRNT DD statements are as follows:

```
//PANSEQ DD DSN=&&X,UNIT=SYSDA,SPACE=(CYL,5),DISP=(,DELETE),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000)
//PANPRNT DD DSN=COM.PAN.REPORTS,UNIT=SYSDA,
//          DISP=(MOD,KEEP),
//          DCB=(RECFM=FB,LRECL=133,BLKSIZE=2660)
//          OR
//PANPRNT DD SYSOUT=A
```

PANDD2 and SYSPUNCH are specified similarly to PANPRNT.

Each PANVALET library identified in the JCL must also be identified to UEDIT by adding a statement to the UEDTB1 table. An example of such a set of definitions follows.

In the Com-plete JCL for a PANVALET library identified by:

```
//MYPANLIB DD DSN=USER.PAN.MYLIB,DISP=SHR
```

the corresponding UEDTB1 table entry would be:

```
CMEDTB1 ID=MP,DSN=MYPANLIB,ACM=PANVALET
```

Note that the ID is the two-character value assigned by the user, and the DSN is the DD name, not the data set name.

The two PANVALET load modules made resident within Com-plete will require additional storage from within the Com-plete region. It will therefore be necessary to increase the REGION parameter for the Com-plete job step. In the case of VS1, the size of the partition in which Com-plete runs must be increased. The increase in size should be the combined sizes of the two load modules.

Step 3: Modify the Com-plete sysparms

The Com-plete sysparms must be modified in order to cause Com-plete to include COMPAM and COMPAN as resident programs. Do this by adding the following statements to the SYSPARM member used to set Com-plete options:

```
RESIDENTPAGE=COMPAM  
RESIDENTPAGE=COMPAN
```



Warning:

Do not modify the COMPAN or COMPAN load modules used by Com-plete while Com-plete is running. During normal use of the PANVALET interface by UEDIT, these modules are refreshed in the Com-plete resident program area. Changing the attributes of these modules while Com-plete is active can have catastrophic results.

Step 4: Add the PANVALET modules to MLPA (MVS and XA only)

Add PVPVLAMS, FGPAN23, and PVEXTUSR to the MLPA list, and IPL if necessary.

Step 5: Name the user exit (optional)

An optional user exit is available that allows users to enforce installation standards for the PANVALET ++ADD and ++UPDATE statements. This exit must be named UXEEX4. Control is passed to UXEEX4 for each of these statements generated by UEDIT or explicitly inserted by the UEDIT user. This routine may be included in the link edit of the UEDIT subprogram UEBP, as the operating system or Com-plete resident program, or in the Com-plete program library. Note that if it is present as a resident program, UXEEX4 must be reentrant.

An example of this exit is provided in the distributed source library COM.SOURCE. See **Security and User Exit Facilities** for more information on the coding and installation of UXEEX4.

Job Entry Subsystem (JES) Interface Modules

The Com-plete Remote Job Entry (RJE) functions and online utility UQ require an interface with the operating system's Job Entry Subsystem (JES) in order to submit jobs and retrieve job queue information. In order to provide modularity and JES independence, all RJE and UQ functions requiring interface with the installation's JES pass through the Com-plete JES Interface Module (JIM). The JIM is a collection of routines used by Com-plete and its utilities to accomplish functions that are dependent upon the installation's JES. A JIM exists for each JES or spooling system supported by Com-plete. Some of the JIMs are distributed in source and load module form, while others are in load module form only.

The Com-plete JIMs, along with the JES systems and operating systems in which they are supported, are listed below.

System	JES2	JES3	VSE/Power	OS
MVS	JESCSERV	JESCSERV		OS/390 2.6 and higher
MVS	JES2CSER	JES3SERV		up tp OS/390 2.10
VSE/ESA			TTJIPOW2/ TTYIPOW3	VSE/ESA 2.1 - 2.4 VSE/ESA 2.5 - 2.6

This chapter covers the following topics:

- MVS JES Interface Modules
- Extended Console Server

MVS JES Interface Modules

SOFTWARE AG recommends to use the common interface module JESCSERV. Access to JES is done by the internally called Entire System Server JES interface XCOMJESC. This one exploits the OS/390 MVS Subsystem Interface functions 79 (SYSOUT API) and 80 (Extended Status) and accesses OS and JES2 dependant controlblocks. It addresses the life-of-job subsystem, i.e.the primary or alternate JES subsystem were Com-plete was initiated. Whenever you are upgrading your OS and/or JES environment please ask SAG support about necessary updates to XCOMJESC. XCOMJESC issues SAF checks for class JESSPOOL. Applymod 10 has no influence on these checks.

The common interface module JESCSERV is delivered as load module. There are no OS or JES dependencies, so you don't need to assemble it. JES dependancies have been moved to the TTJ2MVS (JES2) and TTJ3MVS (JES3). These modules together with TTJIMVS create the UQ A display. Whenever you upgrade your JES, you have to reassemble them using the new JES MACLIBs.

For JES3 additional restrictions apply:

- Input JCL (JESJCLIN) and SYSIN datasets are not selectable.

- Active SYSOUT datasets (not yet closed and freed) are not selectable. This includes the active SYSLOG.
- UPDATE authority is required to access spool datasets.
- MODIFY commands for jobs in DJC networks are currently not possible.

As soon as IBM lifts any of these restrictions in the Subsystem Interface, our JES interfaces will be modified.

If your OS version doesn't meet the prerequisites to use this interface, you can still use the old interfaces JES2SERV/JES3SERV delivered on the source library. For more information see the Installation documentation.

JES3

The MVS JES3 JIM contains no direct interface with the JES3 subsystem. Since Com-plete and the JES3 global processor may be on separate processors, information and commands are passed via an MVS file. The JES3 information is extracted by a Software AG-supplied Dynamic Support Program (DSP) and passed via the MVS file to the JES3 JIM (TTJJES3). The DSP updates the information in the MVS file approximately every 10 to 20 seconds in a normally loaded JES3 system; this time may be longer, however, if the JES3 system is lightly loaded.

This dataset is allocated and initialized during Com-plete installation, and must be defined using the UQJ3JOBS DD card in the Com-plete startup procedure (see the Installation and Migration documentation).

By default, the dataset contains 5 tracks of information. The first 4 tracks are used to pass job information from the DSP to Com-plete. The information tracks must be initialized to contain 32720 byte records. These records contain individual 48 byte entries describing jobs found in the JES3 system, the maximum number of jobs is therefore $(4 * 32720)/48=2726$. If it is required to increase the maximum number of jobs, then the variable NTRACKS in the DSP module IATVQJ3 must be increased and the JES3 interface reinstalled.

The remaining track is used to transfer JES3 operator commands from the Com-plete JES3 interface to the DSP for execution by JES3. The command track consists of 80 byte records and must be initialized to contain the maximum of 80 byte blocks which will fit on the device type on which the dataset is to be placed. Please consult the hardware specifications for the device type to obtain this information.

JES2/JES3 Server Commands (only for the old interfaces JES2SERV and JES3SERV)

Commands can be passed to the JES2/JES3 servers using the SERV operator command of Com-plete. The format for operator commands to the JES subsystems is:

```
SERV server-id,cccc,oooo,oooo
```

where:

SERV	is the Com-plete operator command
server-id	is the JES2/JES3 server name
cccc	is the command to be issued (see below)
oooo	are the optional parameters for the issued command

Available commands are:

STRT,j1,j2...jn	causes the server to initialize control blocks for the JES2/JES3 subsystem(s) as specified in the option parameters j1-jn. Example: SERV JES2,STRT,JES2
STOP,j1,j2...jn	causes the server to terminate the Com-plete server environments of the JES2/JES3 subsystems as specified by option parameters j1-jn. Example: SERV JES2,STOP,JES2
REFR,j1,j2...jn	causes the server to refresh the Com-plete server environments of the JES2/JES3 subsystems as specified by option parameters j1-jn. A refresh causes the JES2/JES3 server to update the server view of the JES environment. This includes closing and deallocating spool datasets which JES no longer uses, and allocating and opening new spool datasets added dynamically to the JES environment. Example: SERV JES2,REFR,JES2
STAT	causes a one-line message to be issued, indicating the status of each of the JES2/JES3 subsystems which this server has initialized. Example: SERV JES2,STAT

Extended Console Server

On systems that support Extended Console the Com-plete Console Server should be used to receive the Console messages. If running in a Sysplex environment this is the only way to receive Console messages since there is no CRWTOTAB any more. The console messages are stored in a table and can be displayed with the UQ M (console display) function. Messages from 1, 2, 3 or all systems in a sysplex may be received according to the server configuration. Also the number of stored messages can be configured (default=512 messages). The Console Server is started automatically at Com-plete startup if a SERVER statement (see below) is encountered in the SYSPARM member. It can also be activated/deactivated dynamically using the SERVER statement in UCTRL.

Syntax:

```
SERVER=(name,TLINCONS,slots,consname,hcset,automsgs,scope1 ,scope2 ,scope3)
```

where:

name	is a unique server name within each copy of Com-plete.
TLINCONS	is the name of the server initialization program.
slots	specifies the number of messages held in the incore table.
consname	is the console name for MCSOPER Macro. It must be unique in the sysplex.
hcset	(Y/N) specifies whether the hardcopy set is to be received by this console.
automsgs	(Y/N) specifies whether messages that can be automated are to be enqueued to this console.
scope1	specifies the name of the first system from which messages are to be received or ALL to receive messages from all systems in the Sysplex. If not specified, only messages from the local system will be received.
scope2,scope3	specifies a second or third system from which messages are to be received. Do not code if scope1 is ALL.

Example:

```
SERVER=(CONSOLE,TLINCONS,2000,COMP51A,Y,Y,DAEF,DAEY)
```

Note:

Hcset=Y is required if outstanding replies are to be displayed. Note that only outstanding replies that arrived after the console was activated can be displayed.

If Automsgs=N is specified, users will not see the system replies to their operator commands.

UDS VSAM SERVICES (MVS Only)

This chapter discusses installation and operation considerations for the UDS VSAM SERVICES. Note that this is relevant for MVS only.

- Installation Considerations
- Operation Considerations

Installation Considerations

UDS invokes IBM's IDCAMS utility program for VSAM SERVICES. The size of IDCAMS with its required subroutines is approximately 300K. Note that this may exceed the normal thread size where UDS is executing. In addition, it takes a considerable amount of time to load these large programs from an MVS load library.

Since IDCAMS and its subroutines are reentrant, they may be placed in the MVS LPA (link pack area). Note that this may decrease your private address space area. Alternatively, you can put the modules in Com-plete's RESIDENTPAGE area. The required control statements for Com-plete's sysparm RESIDENTPAGE are included in the following table.

Note:

If the listed modules are not in the LPA or RESIDENTPAGE area while the UDS VSAM SERVICES are invoked, an S80A abend may occur.

Statement	Approximate Size
RESIDENTPAGE=IDCAMS	124.0K
RESIDENTPAGE=IDCTSEX0	0.9K
RESIDENTPAGE=IDCTSTP0	1.8K
RESIDENTPAGE=IDCRI01	24.0K
RESIDENTPAGE=IDCRILT	0.6K
RESIDENTPAGE=IDCRIKT	0.1K
RESIDENTPAGE=IDCTSRI0	2.8K
RESIDENTPAGE=IDCCDDE	52.0K
RESIDENTPAGE=IDCDE01	36.0K
RESIDENTPAGE=IDCTSTP6	1.6K
RESIDENTPAGE=IDCTSUV0	1.2K
RESIDENTPAGE=IDCCDLC	2.2K
RESIDENTPAGE=IDCLC01	36.7K
RESIDENTPAGE=IDCTSLC0	8.9K
Total	292.8K

Operation Considerations

As a multi-user address space, the Com-plete online environment places some restrictions on the use of the IDCAMS utility program. Since Com-plete controls VSAM I/O operations, it is not possible to access VSAM data sets that are not connected to Com-plete with the REPRO and PRINT commands. In general, however, performance of all LISTCAT, DEFINE, DELETE, and ALTER operations is possible.

UDVS VSAM SERVICES (VSE Only)

This section discusses installation and operation considerations for the UDVS VSAM SERVICES. Note that this is relevant for VSE only.

- Installation Considerations
 - Operation Considerations
-

Installation Considerations

UDVS invokes IBM's IDCAMS utility program for VSAM SERVICES. The size of IDCAMS with its required subroutines is approximately 300K. Note that this may exceed the normal thread size where UDVS is executing. In addition, it takes a considerable amount of time to load these large programs from an VSE load library.

Since IDCAMS and its subroutines are reentrant, they may be placed in the VSE SVA. Note that this may decrease your private address space area. If this is not possible, you must put the modules in Com-plete's RESIDENTPAGE area. The required control statements for Com-plete's sysparm RESIDENTPAGE are included in the following table.

Statement	Approximate Size
RESIDENTPAGE=IDCAMS	30K

Operation Considerations

As a multi-user address space, the Com-plete online environment places some restrictions on the use of the IDCAMS utility program. Since Com-plete controls VSAM I/O operations, it is not possible to access VSAM data sets that are not connected to Com-plete with the REPRO and PRINT commands. In general, however, performance of all LISTCAT, DEFINE, DELETE, and ALTER operations is possible.

UDS VSAM SERVICES (MSP/FACOM Only)

This chapter discusses operation considerations for the UDVS VSAM SERVICES. Note that this is relevant for MSP/FACOM only.

To use the VS function of UDS to access IDCAMS, the following sysparm statements must be added to make the IDCAMS nucleus available to Com-plete utilities without the need to load the programs into thread:

Statement	Value
RESIDENTPAGE	IDCAMS
RESIDENTPAGE	KQCRI01
RESIDENTPAGE	KQCTSRI0

Security Systems

This chapter covers the interfaces to the various Security Systems on the market as well as how they interface with the Com-plete Security System. A good understanding of the relevant system is essential as we can only discuss the Com-plete side of the interfaces here. For the purposes of this description, ACF2, RACF and TOP SECRET are known as "external security systems" and Com-plete Security is known as COMSEC. Com-plete also has an interface to the Natural Security system for logon validation.

This information is provided under the following headings:

- Modes of Operation
 - External Security System Operation
 - Com-plete Security Operation
 - External Security with COMSEC
 - Interface To Natural Security
 - Defining Com-plete to ACF2
 - Defining Com-plete to RACF
 - Defining Com-plete to TOP SECRET
-

Modes of Operation

Through the SECSYS sysparm, you can choose which security system is to be used. The currently available choices are RACF, ACF2, and TOP SECRET, and/or COMSEC.

External Security System Operation

Com-plete uses the System Authorization Facility (SAF) to interface to the external security system. All logons and logoffs are verified/notified via the external security system and access to data sets is also verified through the specified security system via the SAF interface. Com-plete uses this interface as it is common to all systems, therefore, except for a few small exceptions, the sysparm specification is only used where it is necessary to identify the external security system name.

Com-plete identifies itself as the caller via the SUBSYS keyword on the RACROUTE call. The format of the ID is as follows...

```
SAGCOMvr
```

where "vr" is the current Com-plete version release level. This has implications for the various external security systems which are discussed later.

As various users will run within the same address space, the external security system must be aware of that fact (for example, ACF2 must know the Com-plete address space as Multiple User Address Space "MUSASS") otherwise, requests will be given the authority of the actual address space which is normally

too high for the general user.

The Com-plete module issuing the SAF requests runs with AMODE=31 and RMODE=ANY. This means that the ACEEs built by the security can exist above the 16 MB line if the external security system supports this.

Com-plete Security Operation

Com-plete Security holds all information on an Adabas file. When the security system is loaded, the data is stored in tables within the address space to be queried as to the access that a user may have to various resources. Refer to the Com-plete Security documentation for more details on defining access rules.

External Security with COMSEC

The only current duplication between these two is access to data sets and system entry validation. The access calls that are made are made to an internal security routine which then calls the various exits or products installed. The application therefore gets no indication of who fails the request, it simply knows that it failed. In the case where both are specified, one OR the other has the chance to fail the request. To avoid unnecessary duplication, you should allow anybody running under that Com-plete have access to the data sets, while using the other system to actually validate the access.

If an external security system is installed, you will have all user IDs defined there. Therefore, the logon request is ignored by Com-plete Security and issued through the external security system.

Interface To Natural Security

An interface is also available to the Natural Security System. In this case, if the user ID is not defined to Com-plete, Com-plete checks the user ID and password against the Natural Security System file. If found, the user is logged on with the entered user ID and with a model user ID of "SYSNAT" from the Com-plete user ID file. If the user ID exists, but the password check fails and an external security system has been specified at startup, the password violation is ignored and the password is verified via the external security system.

To use this feature of Com-plete, the following steps must be taken.

1. Add a model user ID SYSNAT using the Com-plete user ID maintenance utility.
2. Specify the Com-plete sysparms NATSECDB and NATSECFN to identify the Natural Security System file to be used.

Defining Com-plete to ACF2

As stated previously, Com-plete uses the SAF facility. Therefore, the ACF2 SAF exit must be installed and active on the system. This can be checked by entering the command T C(GSO) followed by LIST OPTS in the TSO ACF command. Options SAF and STC must be specified (they are not specified by default). Please refer to the ACF2 documentation for more information. The user ID assigned to the Com-plete must be defined with the following privileges.

```
JOBFROM MUSASS NO-SMC STC
```

For Job Submission, Com-plete inserts the /*JOBFROM control card in the second line to ensure that the Com-plete supplied card is seen first by ACF2. This means that the user does not need to supply a user ID and password in the JCL and it also means that Com-plete does not have to remember the password to insert it for the user.

ACF2 v. 5.2 and lower:

Assuming that most installations run as advised by the ACF2 installation documentation, the "SAFSAFE" record lets all SAF requests through without checking, as with the following "SAFSAFE" definition.

```
CLASSES(-) CNTLPTS(-) SUBSYS(-)
```

the following "SAFPROT" definition is required for Com-plete:

```
CLASSES(-) CNTLPTS(THREAD-) SUBSYS(SAGCOMvr)
```

Where "vr" is the current version release for Com-plete.

ACF2 v. 6 and above:

The SAF interface in ACF2 has changed. You must use ACF2 conversion facilities to switch your old definitions to match the new version (the old definitions can remain, but will be ignored). The following SAFDEF definition is an example of what is required for Com-plete:

```
SAFDEF COMvr LAST CHANGED BY BHD ON 23/9/97-15:51
  FUNCRET(4) FUNCRSN(0) ID (COMvr) MODE(GLOBAL)
  RACROUTE(SUBSYS=SAGCOMvr REQSTOR=-) RETCODE(4)
```

Defining Com-plete to RACF

As stated previously, Com-plete uses the SAF facility. Standard RACF installation results in the installation of the RACF SAF routines.

No changes to the RACF definition are necessary for Com-plete.

For Job Submission, when the USERID and PASSWORD cards are not provided on the Job Card, Com-plete supplies both cards to identify the job submitter. However, this means that Com-plete must remember the password to insert it for the user, therefore applymod 83 is incompatible with the use of RACF on a system.

Defining Com-plete to TOP SECRET

The following describes the steps required to define Com-plete to a CA-TOP SECRET security system. For the sake of clarity, the following examples do not take advantage of techniques such as the defining of profiles, etc. Where examples are given, you are also referred to the relevant CA-TOP SECRET documentation where you will find more information.

Com-plete is defined as a system facility by CA-TOP SECRET by default. This is described in the TOP SECRET documentation CA-TOP SECRET Implementation: Other Interfaces Guide, section Com-plete. The modifications described in the subsection Required Modifications are no longer necessary, as Com-plete interfaces with TOP-SECRET via the SAF interface as described above. To avoid confusion,

an update request for this section has been issued.

You must change this facility definition using the TSS MODIFY FACILITY command to set the following options:

```
NOABEND          TENV
```

For more information, see the CA-TOP SECRET MVS Control Options Guide.

You now must create an ACID to identify this Com-plete release and link this ACID to the Facility. The name we have chosen for this ACID is the same as that used for the subsystem name we use on the RACROUTE calls documented previously. This is simply as a naming convention and is not necessary for the operation of this interface. You can choose your own names if you so wish. For our example, the ACID we use is SAGCOMvr (where "vr" is the Com-plete Version and Release). The ACID is created with a TSS command as follows:

```
TSS CREATE(SAGCOMvr) TYPE(USER) NAME('COMPLETE vr')
          DEPT(RDDEPT) PASS(NOPW) FAC(ALL) MASTFAC(COMPLETE)
```

For more information, see the CA-TOP SECRET MVS Implementation documentation.

Each Com-plete startup procedure must use the relevant ACID. The ACID that should be used can be identified with the following TSS command:

```
TSS ADD(STC) PROC(pppppppp) ACID(SAGCOMvr)
```

where pppppppp is the procedure name.

For more information see the CA-TOP SECRET MVS Implementation documentation.

Users must then be authorized to logon to Com-plete. This can be done with the following TSS command:

```
TSS ADD(uuuuuuuu) FAC(COMPLETE)
```

where uuuuuu is the user ID.

For more information see the CA-TOP SECRET MVS Implementation documentation.

For Job Submission, Com-plete inserts the constant "x'FF',c'TSS'" followed by the address of the user's ACEE in columns 73-80 of the Job Card. In this way, the user does not have to supply a user ID and password in the JCL and it also means that Com-plete does not have to remember the password to insert it for the user.

The DB2 Interface

The Com-plete/DB2 interface uses the Call Attachment Facility (CAF) of DB2.

With the new dispatcher of Com-plete 5 and above, the internal structure of the DB2 Interface has changed. All CAF processing now runs under control of the same task as the application (usually, Natural) as opposed to the special subtasks used in previous versions of Com-plete.

Since DB2 does not support more than one session per subtask, Software AG recommends that you allocate a separate task group (see **Sysparm TASK-GROUP** in the **Initialization** section of this documentation) with a sufficient number of subtasks in it and run your DB2 applications on this task group.

Note that the number of subtasks in a task group can be changed dynamically using Com-plete operator command TASKS.

During startup, Com-plete is connected to DB2. The DB2 load-library (DSNLOAD) must be concatenated to the COMPINIT/COMPLIB datasets, since all relevant modules will be loaded dynamically.

SYSPARM Considerations

The DB2 interface implementation requires the following SYSPARM definitions:

```
RESIDENTPAGE=DB2COMRE
```

```
SERVER=(DB2,TLINDB2,ssid,plan)
```

where:

ssid	The DB2 subsystem ID for the CAF CONNECT call.
plan	The default plan name if the application does not use an explicit open.

DB2 services for the application are provided via a standard CAF call function:

```
CALL DSNALI and/or CALL DSNHLI
```

This entry point will be resolved by including the module DB2COM from the distributed load library during link edit.

The user application can issue explicit CAF opens by issuing an OPEN call that provides the plan name to be used. The Com-plete/DB2 interface will issue implicit opens, with the plan-name specified in the SERVER sysparm definitions, if the first call for DB2 is not an OPEN call.

Natural

The Natural interface is described under the following headings:

- Installing the Natural Buffer Pool Manager
- Natural Batch

Installing the Natural Buffer Pool Manager

The buffer pool initialization is performed via SERVER definitions in the Com-plete system parameter file (see also the Natural documentation). These definitions must look like this:

```
SERVER=(NATBPSxx,NCFBPSxx,...) for V2.3 and 3.1 buffer pools
```

The server modules must be linked using the following linkage editor commands:

```
INCLUDE NATLIB(NCFBPSnn)
INCLUDE COMLIB(TLINNSRV)
NAME NCFBPSnn(R)
```

where:

- NATLIB is the Natural distribution library.
- COMLIB is the Com-plete release load library.

Linkage editor commands for VSE:

```
// LIBDEF OBJ,SEARCH=(SAGLIB.NATvrs,SAGLIB.COMvrs)
// LIBDEF PHASE,CATALOG=SAGLIB.COMUSER
// OPTION CATAL
PHASE NCFBPSnn,S
INCLUDE NCFBPSnn
INCLUDE TLINNSRV
ENTRY NCFBPSnn
/*
// EXEC LNKEDT,PARM='AMODE=31'
/*
```

The Natural buffer pool initialization modules are loaded dynamically during Com-plete initialization. The linked module must therefore be placed in a load library contained in the COMPLIB concatenation.

Note that the Natural 2.2 buffer pool manager requires COMSTOR to be defined. Failure to do so will result in a user 255 abend when trying to initialize the Natural 2.2 nucleus.

Natural Batch

Natural Batch users who require direct access Com-plete functions (for example, RJE,MESGSW), must define these functions as CSTATIC and link the module COMPBTCH (the Com-plete Batch interface module) to the Natural nucleus. This stops Natural loading the individual modules at runtime (which are typically those for the online environment).

CA-DYNAM from Computer Associates (VSE only)

The CA-DYNAM products use a SYSTEM ADAPTER to trap and analyze the SVCs.

This SYSTEM ADAPTER is normally started via CASAUTIL in the procedure \$0jcl... for the BG partition after IPL.

The CA SYSTEM ADAPTER must be started before our own system interface is initialized by running COMSIP.

Before starting Com-plete, the CA SYSTEM ADAPTER must be disabled for the Com-plete partition. This can be done using

```
// UPSI    00000001
// EXEC    CASAUTIL
DISABLE nn ALL --> where nn is the partition id
/*
```

After Com-plete shutdown, the SYSTEM ADAPTER must be re-enabled:

```
// UPSI    00000001
// EXEC    CASAUTIL
ENABLE nn      --> where nn is the partition id
/*
```

The job control to disable and enable the CA SYSTEM ADAPTER can be included in the Com-plete start job.

IBM Language Environment Considerations

This chapter covers the following topics:

- Saving Thread Storage
 - Receiving IBM Language Environment Runtime Messages and Dumps
-

Saving Thread Storage

When using IBM Language Environment with programs running under Com-plete, place the following statements in the Com-plete SYSPARMs in order to save thread storage:

```
RESIDENTPAGE=CEEBINIT  
RESIDENTPAGE=CEEBLIBM  
RESIDENTPAGE=CEEBLRR  
RESIDENTPAGE=CEEEV005  
RESIDENTPAGE=CEELRRIN  
RESIDENTPAGE=CEELRRTR  
RESIDENTPAGE=CEEMENU3  
RESIDENTPAGE=CEEPLPKA  
RESIDENTPAGE=IGZCLNK  
RESIDENTPAGE=IGZCMGEN  
RESIDENTPAGE=IGZCPAC  
RESIDENTPAGE=IGZCPCO  
RESIDENTPAGE=IGZEINI  
RESIDENTPAGE=IGZETRM
```

Depending on the type of applications running, other routines may be loaded into thread which you also might want to define as resident modules. After running your applications, use UCTRL subfunction OP to find out if there are such routines being loaded, their size and frequency of use.

If your application programs are reentrant, they, too, are candidates for becoming resident.

The IBM Language Environment runtime tends to require approx. 200 Kbytes of working storage in the Com-plete thread below the 16 MB line when the default LE runtime options are in effect. For most applications, this size can be reduced significantly by explicitly changing some of the LE runtime options. Successful tests have been run in threads with just 50 Kbytes below the line for many COBOL programs with the following runtime options in effect (set in CEEUOPT):

```
ALL31=(ON),  
BELOWHEAP=(1K,4K,FREE),  
NONONIPSTACK=(4K,4K,ANYWHERE,KEEP),  
STACK=(32K,32K,ANYWHERE,KEEP),  
STORAGE=(00,NONE,00,8K)
```

Similar options exist for other programming languages in the IBM Language Environment. For more information, please refer to the appropriate IBM documentation. While these settings are likely to work for other applications as well, Software AG cannot guarantee that your applications will run correctly with the same options or in the same thread size. In any case, the resulting thread region size required for each application must be set accordingly in ULIB.

Receiving IBM Language Environment Runtime Messages and Dumps

Each user can choose from three different ways to handle SYSOUT output. The choice is defined by means of the Com-plete user's hardcopy destination:

1. If the hardcopy destination is the name of an existing printer, the output is routed to this printer.
2. If the hardcopy destination is a dummy printer (any name which is not the name of any printer in the network), the output remains in Com-plete's spool file and can be viewed and/or selectively printed using USPOOL. Make sure you have a big spool file and/or specify a short printout expiration time if you choose this option.
3. If a user has no hardcopy destination defined, his output is routed to his terminal after termination of the application.

Under certain circumstances, like abnormal program termination while running with STAE / TRAP(ON) option, IBM Language Environment provides additional problem related information which it tries to write to a file described by DD/DLBL name CEEDUMP.

To get this information when running under Com-plete, add the DD/DLBL name CEEDUMP to the source module ULSODDT1 and re-assemble this module. CEEDUMP output will then be handled for each user individually, in the same way and at the same destination as standard SYSOUT output.

VSE only:

LE/VSE programs can only run in a THREAD-GROUP running in the partition key:

```
THREAD-GROUP=( aaa , ( bbb , nnn , nn , nn , nn , N ) )
```

File Transfer

Com-plete supports 2 methods of file up/downloading from/to PCs:

- UEDIT
 - IND\$FILE (MVS only).
-

UEDIT

READ from / SAVE to Library 'PC' in UEDIT. This method (applicable to SOURCE and OBJECT files) requires ENTIRE CONNECTION to be active on the PC. SOURCE files are converted to ASCII on download and to EBCDIC on upload. Object modules are encrypted on download. These files are unusable in ASCII format but may be sent to a different site via diskette or email. On upload it is decrypted; the original file is restored. A checksum ensures data integrity.

IND\$FILE (MVS only).

This method requires IBM's file transfer program IND\$FILE for TSO and corresponding support on the PC (terminal emulator). In order to make IND\$FILE executable under Com-plete, proceed as follows:

Link-edit module COM\$FILE (supplied in the LOAD distribution data set) with IBM's IND\$FILE program for TSO. Use the following sample JOB:

```
//COMLINK JOB ...
//COMLKD PROC N=,RENT=,NCAL=NCAL
//LKED EXEC PGM=HEWL,
// PARM='LET,&NCAL,&RENT,AMODE=24,RMODE=24'
//SYSPRINT DD SYSOUT=*
//COMPLIB DD DSN=COM.LOAD,DISP=SHR
//LINKLIB DD DSN=SYS1.LINKLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLOAD DD DSN=COM.USER.LOAD(&N),DISP=SHR
//SYSLIN DD DDNAME=SYSIN
// PEND
//IND$FILE EXEC COMLKD,N=IND$FILE,RENT=NORENT
INCLUDE COMPLIB(COM$FILE)
INCLUDE LINKLIB(IND$FILE)
ENTRY COM$FILE
/*
```

Catalogue IND\$FILE using ULIB, specify the options RG=80K and PV. IND\$FILE may not release the thread for the duration of the file transfer, therefore it may make sense to allocate a separate thread group and a separate task group for it in order to avoid delays for other users.

Note:

IND\$FILE may be called only from the COM-PASS menu.

For details on IND\$FILE usage refer to your emulator documentation (file transfer).

Security and User Exit Facilities

This part of the Com-plete System Programming documentation describes the security and user exits available with Com-pete.

This information is organized under the following headings:

- Introduction
- User Exit Considerations by Type of Exit
- Creating or Modifying a User Exit
- ACSUUEX1 - ACCESS Write-Intercept Exit
- ACSUUEX2 - ACCESS Read-Intercept Exit
- SDAMSEX1 - SDAM API Security Exit
- TUDUEX1 - Select Dumps by User-Defined Criteria
- UCOEX1 - UCOPY User Exit
- UDMPX1 - UDUMP Security Exit
- UDSEX1 - UDS Security Exit (MVS Only)
- UDVSX0 - Usage Control of UDS/UDVS VSAM SERVICES
- UDYEX1 - Control Dynamic Allocation/Deallocation of Datasets using UDDYN (MVS only)
- UEDTB1 - Library Code Table
- ULHMX1 - Hello Message Exit
- ULMSBTCH - Batch Output User Exit
- ULMSDISK - Dynamic Printer Allocation User Exit
- ULINUSER - Com-plete Initialization Exit
- ULOGX1 - ULOG Security Exit
- ULOPADAB - Adabas User Exit
- ULSRMPEX - Modify PF Key Codes
- ULSRPSFS - User-Written Service Routine
- ULSRRJE - Remote Job Entry User Exit
- ULSRSEC - User-Written Service Routine
- UMSEX1 - UM Security Exit
- USTKX1 - USTACK User Exit
- USTRE1 - USTOR User Exit
- UTMEX1 - Timer User Exit
- UTMEX2 - Timer Monitor User Exit

- UTMEX3 - Timer Monitor RJE Exit
- UUEDEX - UED Security Exit
- UUMAX1 - UMAP Initialization Exit
- UUMAX2 - UMAP Command Exit
- UUMAX3 - UMAP Termination Exit
- UUPDX1 - UPDS Security Exit (MVS Only)
- UUQEX1 - UQ Security Exit
- UUSEX1 - USDLIB Security Exit
- UUSPL0 - USPOOL Command Exit
- UUSVX1 - USERV Security Exit (VSE Only)
- UUTEX1 - UTIL Security Exit
- UXEEX1 - UEDIT Initialization Exit
- UXEEX2 - UEDIT Command/Termination Exit
- UXEEX3 - UEDIT RJE Exit
- UXEEX4 - UEDIT LIBRARIAN/PANVALET Exit
- UXEEX5 - Locate Exit

Introduction

This chapter covers the following topics:

- Summary of Available Exits
- Areas of Exit Usage
- ULOG ON Security
- SYSCOM,SYSNAT
- Batch/TPF User IDs
- ULOGX1 Exit
- Program, SD File, File I/O Security
- Control Programs
- Message Switching and Printout Spooling
- Utility and Application Security
- ACCESS User Exits

Summary of Available Exits

The following table summarizes Com-plete's security and user exit facilities. Each of the facilities listed below is discussed later in this chapter.

Facility	Summary
ACSUUEX1	Intercepts all screen data that would normally be sent to the user's terminal for manipulation.*
ACSUUEX2	Passes data on to the target system.*
SDAMSEX1	Controls the use of SDAM API functions.
TUDUEX1	Allows user-defined tests and restrictions for use of the TUDUMP program.
UCOEX1	UCOPY exit. Allows user alteration of the destination supplied.
UDMPX1	Defines security restrictions on the use of UDUMP to control the access of dumps in the online dump library.
UDSEX1	Defines security restrictions on the use of UDS (MVS only).
UDVXS0	Controls/restricts the use of UDS/UDVS VSAM SERVICES.
UDYEX1	Control dynamic allocation/Deallocation of datasets using UDDYN (MVS only).
UEDTB1	Defines the library identification codes used to refer to user libraries.

Facility	Summary
ULHMX1	Allows user alteration or suppression of Com-plete's hello message, based on TID.
ULMSBTCH	Allows modification of parameters used by batch spool output routines to generate DYNALLOC/SEGMENT calls.
ULMSDISK	Allows modification of a dynamically allocated printer TIB entry by a user.
ULINUSER	Allows user processing during Com-plete initialization.
ULOGX1	Defines security restrictions on the use of ULOG prior to various events.
ULOPADAB	Allows user examination and/or alteration of Adabas call parameters.
ULSRMPEX	Allows modification of PF key codes in a MRCB.
ULSRSEC	Controls use of specific application functions, programs, modules, and Com-plete utility programs.
ULSRRJE	Examines and/or modifies the (RJE) input data submitted for background processing via the RJE function call.
ULSRSEC	Controls access to files.
UMSEX1	Controls the use of the message switching functions.
USTKX1	
USTRE1	Defines security restrictions on the use of operator command functions.
UTMEX1	Examines each new timer request to be added to the timer SD file by UTIMER.
UTMEX2	Is a timer monitor exit called every minute and for each request that is to be served.
UTMEX3	Is a timer monitor exit called at RJE job submission.
UUEDEX	Defines security restrictions on the use of UED.
UUMAX1	Defines security restrictions on the use of UMAP, and allows user modification of the global defaults to UMAP.
UUMAX2	Defines security restrictions on the use of the UMAP command functions.
UUMAX3	Defines security restrictions on the use of UMAP, forces the cleanup of SD files, and controls the termination of UMAP.
UUPDX1	Defines security restrictions on the use of UPDS (MVS only).
UUQEX1	Defines security restrictions on the use of UQ.
UUSEX1	Defines security restrictions on the use of USDLIB to control the access of SD files.
UUSPL0	Is the USPOOL command exit.
UUSVX1	Defines security restrictions on the use of USER (VSE only).
UTMEX1	Defines security restrictions on the use of UUTIL functions.

Facility	Summary
UXEEX1	Defines security restrictions on the use of UEDIT and user ID access to specific libraries or members.
UXEEX2	Defines security restrictions on the use of UEDIT commands and UEDIT termination.
UXEEX3	Defines security restrictions on the use of UEDIT, and controls the job control conventions for submitted jobs from UEDIT and UPDS. (It is recommended that ULSRRJE be used for this function.)
UXEEX4	Defines security restrictions on the use of UEDIT and the interfaces to PANVALET and LIBRARIAN.
UXEEX5	Examines each LOCATE request to catalog management and allows the recall of migrated data sets.

* These exits only exist on the "host" side of an access request, for example the CICS from which the user is accessing.

Areas of Exit Usage

The Com-plete security and user exit facilities allow you to set restrictions on individuals or groups of individuals (departments) for accessing the various facilities, programs, and functions of Com-plete. Restrictions that can be imposed include:

- Requiring a password in order to gain access to the system;
- Disallowing specific users or groups from using certain application programs or Com-plete utilities;
- Disallowing specific users or groups from using various functions within an application program, while simultaneously allowing the use of other functions within the same program (for example, allowing the display of records for an application file, but disallowing updates);
- Disallowing specific users or groups from using certain control functions provided by some of the Com-plete online utility programs (for example, the K function of the UQ utility program);
- Restricting usage of the UQ functions H, R, C, DE, and S to terminal users who have submitted the specific job being accessed or to users who belong to a specific department;
- Disallowing specific users or groups from viewing certain messages or printout spooling data sets.

In addition to the security checks that can be imposed, an installation can also restrict program execution to specific threads based upon their thread-lock number. For example, if a program is thread-locked to thread two, the installation may choose to force execution to thread one without recataloging the program. This restriction can be forced by program name, thread number, or both.

The areas where security can be defined in order to accomplish these objectives include:

- ULOG ON security;

- Program security;
- SD file security;
- File I/O security;
- Thread-scheduling control;
- Program timing (MVS only);
- Control user IDs;
- Message switching security;
- Printout spooling security;
- Utility security;
- Application security;
- Data manipulation using ACCESS.

Many of these security areas allow you to define and write subroutines to meet the needs of your installation. Their general purpose and function is presented in the following sections.

ULOG ON Security

Before accessing any application program or Com-plete utility, a terminal operator can be required to enter a "*ULOG ON" request as identification to Com-plete security and accounting routines. This logon requirement is an optional feature of Com-plete selected at initialization time (see the ACCOUNTING and PASSWORD sysparms).

When a user logs on, a user ID and a password, must be supplied. Com-plete verifies the information by invoking an external security package (for example, RACF) via the SAF interface, or by accessing the Com-plete system data set, which contains user ID information. For each user allowed to access the Com-plete system, this information consists of the following:

- A unique user ID that identifies the user;
- The account number (or group number) with which the user ID is associated;
- The password for the user ID;
- The control status to be assigned to the user ID. This status is used by various Com-plete utilities to restrict the use of certain privileged functions (e.g., the K function of the UQ utility);
- The authorization code for the user ID;
- The sending and receiving message and printout class codes for the user ID.

Each time a "*ULOG ON" request is entered, the user ID is accessed in the system data set and the password, if required, is validated. If the user ID is not found, or if the password is not verified, the ULOG ON request is aborted.

If the user ID is found and the password is verified, a control block for the user containing information from the user ID record is created, plus additional room for keeping track of the resources to be assigned to the user ID. This control block is called the user accounting block (UAB), see also the chapter Accounting.

If an external security system such as RACF, ACF2 or TOP SECRET is installed and specified in the Com-plete startup procedure, the user ID/password combination must pass the external security system verification rules. If the user ID/password combination is unknown, the request is rejected with the message returned from the security system.

SYSCOM,SYSNAT

If the user ID is not defined on the Com-plete system data set and the NATURAL Security Interface is active (sysparms with prefix NATSEC), the user ID and password combination is verified against the NATURAL Security File. If the user ID exists and the password is correct, the user ID is logged on with the entered user ID, and the Com-plete required information retrieved using the user ID model SYSNAT.

If the user ID is not defined to the Com-plete system data set, and the NATURAL Security interface is either not installed or rejects the logon, and APPLYMOD 57 is specified, the user ID is logged on irrespective of the specified password and given the Com-plete required information from the user ID model SYSCOM.

In both cases the model user ID must be defined to Com-plete. If an external security system is present, then the model user ID is also validated with the security system.

If the model user ID is defined, then the logon is allowed using the authorization obtained for the model user ID.

If the model user ID is not defined, then the logon is still allowed but no authorization is supplied, that is, any attempted access to items which would normally be protected by the external security system will fail.

Batch/TPF User IDs

For Batch and TPF users, a model user ID is always supplied by the host system in the logon data (SYSBAT and SYSTPF respectively). The logon then proceeds as above, the user IDs SYSBAT and SYSTPF must be defined on the Com-plete system data set and can also be defined to an external security system.

ULOGX1 Exit

In addition to Com-plete security, a user-written security exit, ULOGX1, is called to allow the installation to perform further security checking against passwords, time-of-day, etc. This routine can also change some items in the Logon Information Block that is assigned to the user ID. These items include:

- Control status ;
- Authorization code;

- Account number;
- Message class codes;
- COM-PASS model.

The ULOGX1 security exit is discussed later in this chapter.

Program, SD File, File I/O Security

The user-written security routine ULSRPSFS is called to allow security checks to be issued for any or all Com-plete application functions.

ULSRPSFS either allows or disallows the request by setting a return code. If the request is disallowed, the application program is abnormally terminated.

Control Programs

Some of the functions afforded by the Com-plete utilities are restricted to user IDs that are assigned control status. These functions are listed in the following table:

Utility	unction
UDD	All commands
UDZAP	All commands
ULIB	CAT and DEL commands for PV programs
UM	SCLASS command RCLASS command PURGE command, using TID parameter* DELETE command, using TID parameter* ALT command, using TID parameter* RESET command, using TID parameter*
UQ	K command (see APPLYMOD=6 in Binary Modifications (APPLYMODS))
USTOR	All functions
UUTIL	For authorizations of UUTIL functions, see the Com-plete Utilities documentation

* See APPLYMOD=7 in Binary Modifications (APPLYMODS)

These utility programs and their privileged functions are discussed in more detail in the Com-plete Utilities documentation.

Message Switching and Printout Spooling

Each time an application program or Com-plete utility program makes a request to send a message or printout, Com-plete performs a security check to see if the class codes assigned to the message or printout correspond to the class codes assigned to the user ID using the sending terminal. If not, the message switch or printout request is aborted.

Com-plete also checks the class codes for the user ID of the receiving terminal. If they do not match the class codes of the message or printout, the message or printout is aborted.

If an attempt is made to send a message or printout to a terminal not in use, the default class codes for that terminal are used. This allows for sending messages and printouts to terminals to which no one has logged on (e.g., a 3286 line printer).

Default message switching and printout spooling class codes are normally set when TIBTAB is created and/or the user ID is defined. The creation of TIBTAB is defined in the chapter entitled TIBTAB - Terminal Information Block Table. User IDs are defined using the UUTIL utility program, which is described in the Com-plete Utilities documentation.

Utility and Application Security

Utility Security

Most of the Com-plete utilities described in the Com-plete Utilities documentation contain at least one exit point for exiting to a user-written routine. Each of these routines' functions and linkage conventions is discussed later in this chapter. These routines allow you to define your own security restrictions on the use of the utility programs.

In addition, the UQ utility program recognizes certain job control comment cards that can be used to restrict usage of the UQ functions H, R, C, DE, and S. These statements are:

xxxUQ USER ID	restricts user IDs
xxxUQ ACCOUNT	restricts account numbers
xxxUQ AUTHORIZE	restricts authorization codes
xxxUQ DISALLOW	restricts all access
xxxUQ ALLOW	removes access restrictions
xxxUQ USER	passes information to the exit

where xxx is "/*" in MVS, and "*" in VSE.

Note that these job control comment statements must exist in the job stream preceding the first EXEC statement. Further information on their usage can be found in the Com-plete Utilities documentation.

Generally, if more than one of the comment statements are present, only one condition must be met to pass security checking. If none are present, UQ either disallows everyone using the indicated functions or allow everyone, depending upon the default Com-plete sysparm UQDEFAULT.

Application Security

If an application program needs to restrict some but not all of its functions, the GETCHR function can be used to establish the required authorization.

The GETCHR function enables the program to determine the user ID, account number, authorization code, and control status of the terminal user. This way, some functions can be restricted to certain user IDs.

The GETCHR function is fully described in the Com-plete Application Programmer's documentation.

COM-PASS Security System

All or parts of utility and application security can be accomplished through the use of the COM-PASS security system.

COM-PASS users are defined by a User Profile. A part of this profile is the User Transaction Profile. Every access the user attempts to make to a transaction is checked by COM-PASS. If the user is not allowed to use the transaction, either of the following messages are displayed:

```
OVL0007 (-) - ACCESS TO REQUESTED PROGRAM DISALLOWED
UMP0023 - SECURITY VIOLATION - ACCESS TO PROGRAM DISALLOWED
```

The COM-PASS User Profile is determined by:

- The COM-PASS security indicator. This defines the user to be checked by COM-PASS for transaction security within Com-plete.
- The COM-PASS menu transactions (A through I). These are the transactions that are defined for the user, which appear on the COM-PASS main menu as service programs.
- The COM-PASS startup program. This can be set to any valid COM-PASS, Com-plete, or user transaction program. To have the main menu appear first, set this parameter to USTACK.

The user is not allowed to change the transaction authority. When the security switch is set, the user is only allowed to use the defined transaction programs that appear on his/her main menu as service programs.

ACCESS User Exits

ACCESS provides user exits that enable user-written routines to be called by the ACCESS transaction. These user exit routines can examine and alter data at the following times:

- Before writing the data from the target node to the host terminal;
- Before the terminal input is sent to the target node.

Note:

All references to "target node" refer (currently) to a Com-plete system.

The names of these exits are ACSUUEX1 (writing) and ACSUUEX2 (reading). Each one is explained later in this section

User Exit Considerations by Type of Exit

Each Com-plete user exit is classified as a batch, thread, or nucleus exit. Each type of exit has unique attributes and considerations. The conventions table for each exit specifies the type of exit.

This chapter covers the following topics:

- Batch Exits
 - Thread Exits
 - Nucleus Exits
-

Batch Exits

Batch exits are called from Com-plete batch utility programs. The exit can perform most operation system functions available to a batch program. Note that batch exits should not perform any input/output functions on files defined for that utility.

Note:

MVS linkage editor overlays are not permitted.

Abnormal termination within a batch exit will cause the Com-plete batch utility to terminate abnormally.

Thread Exits

Thread exits, called from Com-plete online utilities, execute as extensions of these utilities. The exit is restricted to using storage defined when the online utilities were cataloged to Com-plete. Thread exits are called in the PSW-protect key of the thread.

Note:

MVS linkage editor overlays are not permitted.

Thread exits can use Com-plete functions defined in the Com-plete Application Programmer's documentation. Storage can be acquired only from the region of the thread defined when the online utility was cataloged. GETMAIN/GETVIS requests are satisfied from the thread's region. The online utility may need to be recataloged with a larger region size.

If the exit is thread locked, its caller must also be thread locked.

A thread exit can also be used by specifying the RESIDENTPAGE sysparm.

Note that exits placed in either resident area must be reentrant.

When exits are specified as being RESIDENTPAGE, the exit is entered in the AMODE it was linked with for the operating systems. In this way, thread exits can be run in 31 bit mode and can also reside above the 16 MB line. This means they can be linked with RMODE=ANY. Exits must always return to the caller in the addressing mode they were called in. When the exit is called, register 14 will have the caller's AMODE set as standard.

Abnormal terminations within a thread exit cause the online utility to terminate abnormally.

Nucleus Exits

Nucleus exits, called from the Com-plete nucleus, execute as an extension of that nucleus. GETMAIN/GETVIS requests are satisfied from the Com-plete partition/address space, not from the thread. The nucleus exits execute in the PSW-protect key of Com-plete and in supervisor state. If the exit must refer to areas within a thread, those areas will be in the protect key of the thread during the time the exit has control.

Note that nucleus exits must be reentrant.

Note:

MVS linkage editor overlays are not permitted.

Nucleus exits must not use any Com-plete function defined in the Com-plete Application Programmer's documentation.

Nucleus exits are loaded dynamically. Refer to the description of each nucleus exit in the following sections for additional information.

As nucleus exits are now loaded using standard MVS / VSE loads, on XA and ESA systems the modules are loaded and entered according to their AMODE and RMODE specification. Please note that exits must return to the caller in the caller's AMODE. This is set as standard in the high-order byte of register 14 when the exit is initially called. Return is correct if the following instruction is used:

```
BSM R0,R14
```

This tells the machine to return to the address in R14 while setting AMODE 31 if the high-order bit is on, or AMODE 24 if the high-order bit is off. This will have been set by the caller of the exit.

Abnormal terminations within a nucleus exit cause the Com-plete nucleus to terminate abnormally.

Note that when a nucleus exit routine gets control, the storage key of the user program area of the thread may be in a different key to Com-plete's key. If the exit needs to change the contents of any fields in the user program area, they must do this bearing this fact in mind. The various nucleus exits which may be affected by this have appropriate notes in the Considerations sections of their individual description in this chapter.

Creating or Modifying a User Exit

To create or modify a user exit, follow the procedure below:

Step 1

Ensure that a backup copy is made of the exit and of Com-plete or the affected utility. (Errors in user exits may render Com-plete or the utility unusable.)

Step 2

Modify the current source of exit or the sample in the distributed source library.

Step 3

Assemble and link edit the user exit as follows:

MVS/FACOM Assemble and link edit the user exit into COM.USER.LOAD. If the exit is to be link edited with a utility, then link edit the affected routines using the portion of JCL in \$LINKxx for those routines, where xx represents MVS or F4 as appropriate to your operating system.

VSE: Assemble and catalog the user exit into SAGLIB.COMUSER. If the exit is to be link edited with a utility, then link edit the affected routines using the portion of job control in JCLLINKS.J for those routines. If the exit is loaded dynamically, link edit the user exit into SAGLIB.COMUSER.

Step 4

Make the user exit/utility available to the terminal user by either:

- refreshing the user exit with ULIB if the user exit is loaded dynamically and Com-plete is currently active. Note that the region size parameter of the utility may need to be increased using ULIB in order to accommodate the exit. If the thread is too small to contain the online utility and the exit, an abnormal termination will result; or:
- refreshing the utility with ULIB if the user exit is linkedited with the online utility and Com-plete is currently active. Note that the region size parameter of the utility may need to be increased; or:
- adding the user exit with the RESIDENTPAGE sysparm. RESIDENTPAGE routines save storage that would be needed if the user exit executed in the thread. Note that RESIDENTPAGE routines must be reentrant.

Note:

If the exit is a nucleus exit, Com-plete must be terminated and then reactivated.

Step 5

Test the function of the exit.

ACSUUEX1 - ACCESS Write-Intercept Exit

ACSUUEX1 is a user-written routine that can intercept all screen data that would normally be sent to the user's terminal and modify or delete it as required. It can also generate additional screen data. If a screen is deleted (this means that the data prepared by the target system is not sent to the host terminal), the user-exit routine ACSUUEX1 can invoke the target system's read routine as if the user had viewed the screen, typed in data, and pressed ENTER.

This chapter covers the following topics:

- How to Create ACSUUEX1
 - How to Use ACSUUEX1
 - ACSUUEX1 Conventions
-

How to Create ACSUUEX1

To create an ACSUUEX1 routine, the following steps are required:

Step 1

Code the ACSUUEX1 routine entry of ACSEXITS (A.ACSEXITS for VSE, or ACSEXITS ASSEMBLE for CMS).

Step 2

Assemble and link ACSEXITS (A.ACSEXITS for VSE) into the ACCESS load library (relo library for VSE). Refer to the ACCESS source library during the assemblies for MVS and VSE environments. Or for CMS only, assemble ACSEXITS with the ACCESS macro library referenced.

Step 3

Link edit ACSEXITS (A.ACSEXITS for VSE) to the ACCESS driver for the host Adabas TPF, CICS, or TSO systems. Member JCLLINKA (A.JCLLINKA for VSE) in the ACCESS source library contains the link edit JCL for all TP monitors supported. Or for CMS only, execute the EXEC GENACS.

How to Use ACSUUEX1

Members CCACSWK and CCACSPFX in the distributed source library are DSECTs referred to in the main ACCESS routine. These areas are addressable in the ACSUUEX1 exit. A sample entry routine is provided in member ACSEXITS (A.ACSEXITS for VSE) of the ACCESS source library.

ACSUUEX1 Conventions

The following table summarizes the ACSUUEX1 linkage conventions.

Feature	Convention																
Attributes	None required.																
Size	Restricted to the ACS driver region.																
Registers atEntry	<table> <tr> <td>Register 2</td> <td>Output length (halfword)</td> </tr> <tr> <td>Register 3</td> <td>Address of the output area</td> </tr> <tr> <td>Register 6</td> <td>Address of the ACCESS prefix</td> </tr> <tr> <td>Register 9</td> <td>Address of the ACCESS work area</td> </tr> <tr> <td>Register 10</td> <td>Main ACCESS driver base address</td> </tr> <tr> <td>Register 13</td> <td>Address of an 18-fullword save area</td> </tr> <tr> <td>Register 14</td> <td>Return address within the ACS driver</td> </tr> <tr> <td>Register 15</td> <td>Entry address within ACSUUEX1</td> </tr> </table>	Register 2	Output length (halfword)	Register 3	Address of the output area	Register 6	Address of the ACCESS prefix	Register 9	Address of the ACCESS work area	Register 10	Main ACCESS driver base address	Register 13	Address of an 18-fullword save area	Register 14	Return address within the ACS driver	Register 15	Entry address within ACSUUEX1
Register 2	Output length (halfword)																
Register 3	Address of the output area																
Register 6	Address of the ACCESS prefix																
Register 9	Address of the ACCESS work area																
Register 10	Main ACCESS driver base address																
Register 13	Address of an 18-fullword save area																
Register 14	Return address within the ACS driver																
Register 15	Entry address within ACSUUEX1																
Registers at Return	Registers must be restored, except register 15, which must contain a return code.																
Return Codes	<table> <tr> <td>0</td> <td>Continue with write.</td> </tr> <tr> <td>4</td> <td>Write message and read normally.</td> </tr> <tr> <td>8</td> <td>No write and read. Return.</td> </tr> </table>	0	Continue with write.	4	Write message and read normally.	8	No write and read. Return.										
0	Continue with write.																
4	Write message and read normally.																
8	No write and read. Return.																
Considerations	Must be assembled and link edited with the ACS driver.																

ACSUUEX2 - ACCESS Read-Intercept Exit

ACSUUEX2 is a user-written routine that normally passes the data on to the target system; it can, however, also bypass the target system and invoke ACSUUEX1, the write-intercept exit, in order to produce a response to the user's input. The routine ACSUUEX2 can also be used to initiate commands to the host operating system, for example, CMS/CP commands.

This chapter covers the following topics:

- How to Create ACSUUEX2
 - How to Use ACSUUEX2
 - ACSUUEX2 Conventions
-

How to Create ACSUUEX2

To create a ACSUUEX2 routine, the following steps are required:

Step 1

Code the ACSUUEX2 routine entry of ACSEXITS (A.ACSEXITS for VSE, or ACSEXITS ASSEMBLE for CMS).

Step 2

Assemble and link ACSEXITS (A.ACSEXITS for VSE) into the ACCESS load library (relo library for VSE). For MVS and VSE, have the ACCESS source library referred to during assemblies. Or for CMS only, assemble ACSEXITS with the ACCESS macro library referenced.

Step 3

Link edit ACSEXITS (A.ACSEXITS for VSE) to the ACCESS driver for the host Adabas TPF, CICS, or TSO system. Member JCLLINKA (A.JCLLINKA for VSE) in the ACCESS source library contains the link edit JCL for all TP monitors supported. Or for CMS only, execute the EXEC GENACS.

How to Use ACSUUEX2

Members CCACSWK and CCACSPFX in the distributed source library are DSECTS referred to in the main ACCESS routine. These areas are addressable in the ACSUUEX2 exit. A skeleton entry is provided in member ACSEXITS (A.ACSEXITS for VSE) of the ACCESS source library.

ACSUUEX2 Conventions

The following table summarizes the ACSUUEX2 linkage conventions.

Feature	Convention
Attributes	None required.
Size	Restricted to the ACS driver region.
Registers at Entry	<p>Register 0 Input length.</p> <p>Register 1 Address of the input area</p> <p>Register 6 Address of the ACCESS prefix</p> <p>Register 9 Address of the ACCESS work area</p> <p>Register 10 Main ACCESS driver base address</p> <p>Register 13 Address of an 18-fullword save area</p> <p>Register 15 Entry address within ACSUUEX2</p>
Registers at return	All registers must be restored.
Return Codes	<p>0 Continue with read.</p> <p>4 Tell target node to rewrite the screen.</p>
Considerations	Must be assembled and link edited with the ACS driver.

SDAMSEX1 - SDAM API Security Exit

SDAMSEX1 is a user-written routine called by the SDAM API before a request is passed to SDAM. The exit can be used to impose security restrictions on the use of SDAM functions and resources, over and above those defined in Com-plete.

This chapter covers the following topics:

- SDAMSEX1 Conventions

SDAMSEX1 Conventions

The following table summarizes the SDAMSEX1 linkage conventions:

Feature	Convention
Registers at Return	Registers 12 through 14 must not be modified.
Parameters	Register 1: Points to the SACB (SDAM control block)
Return Codes	0 Allow the function 4 Disallow the function.
Considerations	a Called by SDAM b Linkage is dynamic, with COLOAD

TUDUEX1 - Select Dumps by User-Defined Criteria

In addition to being able to define the JCL PARM string, you can also define your own criteria to select dumps from the dump SD-file for print by using the TUDUEX1 exit.

TUDUEX1 is a user-written service routine that can be used to implement user-defined tests and restrictions when printing dumps out of the Com-plete dump file.

When Com-plete is initially installed, a dummy TUDUEX1 module exists as a member of the distribution source library to serve as a guide. This routine must be coded, assembled, and link edited with the Com-plete batch utility TUDUMP. The initial TUDUMP program contains a dummy TUDUEX1 routine that loads register 3, resets register F to zero, and expands the name of the dumped program.

This chapter covers the following topics:

- How To Use TUDUEX1
 - TUDUEX1 Conventions
-

How To Use TUDUEX1

TUDUEX1 is called after each dump header record is read by TUDUMP and checked according to the key values input with the JCL PARM string. This record contains the UPCB of the dumped program. When TUDUEX1 gets control, register 1 points to the UPCB. An LR R3,R1 instruction is performed at start of TUDUEX1, using R3 as base register for the UPCB DSECT. RF is set to zero, and a branch to subroutine NAMEXP is made in order to expand the name of the dumped program. Your tests and restrictions referring to the UPCB data can now be inserted. For the layout of this DSECT, see CCUPCB in the distributed source library.

TUDUEX1 can provide return code 0 to allow printing of dump, return code 4 to disallow the dump to be printed, or can abend to terminate TUDUMP.

TUDUEX1 Conventions

The following table summarizes the TUDUEX1 linkage conventions.

Feature	Convention								
Attributes	None required.								
Type	Batch.								
Size	No restriction .								
Registers at Entry	<table> <tr> <td data-bbox="423 369 553 411">Register 1</td> <td data-bbox="626 369 1146 443">Points to the UP CB of the dumped program selected for print</td> </tr> <tr> <td data-bbox="423 464 553 506">Register 13</td> <td data-bbox="626 464 1170 537">Address of an 18-word MVS-compatible save area</td> </tr> <tr> <td data-bbox="423 558 553 600">Register 14</td> <td data-bbox="626 558 805 600">Return address</td> </tr> <tr> <td data-bbox="423 621 553 663">Register 15</td> <td data-bbox="626 621 854 663">Entry point address</td> </tr> </table>	Register 1	Points to the UP CB of the dumped program selected for print	Register 13	Address of an 18-word MVS-compatible save area	Register 14	Return address	Register 15	Entry point address
Register 1	Points to the UP CB of the dumped program selected for print								
Register 13	Address of an 18-word MVS-compatible save area								
Register 14	Return address								
Register 15	Entry point address								
Registers at Return	Registers must be unchanged except register 15, which contains the return code.								
Return Codes	<table> <tr> <td data-bbox="423 772 448 814">0</td> <td data-bbox="634 772 821 814">Print this dump.</td> </tr> <tr> <td data-bbox="423 835 448 877">4</td> <td data-bbox="634 835 1162 909">Skip this dump; continue processing with the next dump.</td> </tr> </table>	0	Print this dump.	4	Skip this dump; continue processing with the next dump.				
0	Print this dump.								
4	Skip this dump; continue processing with the next dump.								
Considerations	Must be link edited with TUDUMP.								

UCOEX1 - UCOPY User Exit

UCOEX1 is a user-written routine called by UCOPY, the Com-plete screen to hard-copy utility, before writing the screen-data to the printout spooling system. Possible uses of this exit are:

- Rejection of the request;
- Providing a default hardcopy destination..

This chapter covers the following topics:

- How to use UCOEX1
 - UCOEX1 Conventions
-

How to use UCOEX1

Upon entry to UCOEX1, a set of parameters is received in the form of fullword addresses pointed to by register 1.

Word one of this parameter list contains the address of a full-word to be initialized by the exit; a value of 0 allows access, a value of 4 disallows access.

Word two of this parameter list contains the address of a eight byte field where the exit can supply a valid destination. This value must be in a format acceptable to a PSOPEN call (see the Com-plete Application Programmer's documentation). If this value is set to blanks, the standard default is taken.

Word three of this parameter list contains the address of the buffer containing the screen-data. The exit must only examine this data, any attempt to modify the contents will result in undefined errors.

UCOEX1 Conventions

The following table summarizes the UCOEX1 linkage conventions.

Feature	Convention
Attributes	None required
Registers Entry	Register 0 Register 1 Address of the parameter list Register 13 Address of an 18-fullword savearea Register 14 Return address. Register 15 Entry address.
Registers at Return	All registers must remain unchanged
Return Codes	0 Allow access 4 Disallow access

UDMPX1 - UDUMP Security Exit

UDMPX1 is a user-written routine called by the UDUMP utility program after syntax checking the terminal user's request but before calling the utility itself. This routine allows the user to define security restrictions on the access of dumps in the online dump library.

Because the UDMPX1 module is only loaded once per invocation of UDUMP, internal switches can be set and referenced.

A sample UDMPX1 module is distributed with the Com-plete system as a member of the distribution source library and the distribution load library.

Note:

No security exists for UDUMP functions, unless it is established by you.

This chapter covers the following topics:

- How to Use UDMPX1
 - UDMPX1 Conventions
-

How to Use UDMPX1

Upon entry to UDMPX1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 contains the address of the program name for the dump being accessed. Word 2 of the parameter list contains the address of a return code area in which the status of the request is to be indicated.

To define security, check the program name being passed, establish the desired level of authorization, and set the return code to indicate acceptance or rejection.

Upon return from UDMPX1, the return code area is examined by UDUMP. If the return code is not zero, the operation is aborted and a security violation message is issued.

UDMPX1 Conventions

The following table summarizes the UDMPX1 linkage conventions.

Feature	Convention
Attributes	None required.
Type	Thread.
Size	Restricted to UDUMP region size.
Registers at Entry	Register 1 Address of the parameter list Register 13 Address of an 18-fullword save area Register 14 Return address in the calling module Register 15 Entry address of UDMPX1
Registers at Return	All registers must be unchanged.
Parameters	Word 1 Address of the six-byte program name Word 2 Address of a return code halfword
Return Codes	0 Allow the request. 4 Security violation.
Considerations	a Is loaded once per call of UDUMP. b Is loaded before invoking utility functions.

UDSEX1 - UDS Security Exit (MVS Only)

UDSEX1 is a user-written routine called by the UDS utility program before any command entered by the terminal operator is executed. This module allows you to define security restrictions on the use of the various functions.

The UDS utility program is written as a set of logically related modules, each of which services a specific function. Each function requested by a terminal operator is logically processed by a separate module. In turn, each of these modules issues a call to UDSEX1 before servicing the requested function. Consequently, you can restrict, permit, or eliminate any or all of the UDS functions.

Because the UDSEX1 module is only loaded once per invocation of UDS, internal switches can be set and subsequently referenced. Each new invocation of UDS will load a new version of UDSEX1, causing the switches to be reset.

A dummy UDSEX1 module is distributed with the Com-plete system as a member of the distribution source library and the distribution load library.

Note:

No security exists for UDS functions unless established by you.

This chapter covers the following topics:

- How to Use UDSEX1
 - UDSEX1 Conventions
-

How to Use UDSEX1

Upon entry to UDSEX1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of the 4-byte operation initiating the request. Word 2 contains the address of the file upon which the function will be performed.

Word 3 contains the address of a parameter list that identifies the volume(s) on which the file being processed is located. This parameter list is normally 14 bytes long. If the file resides on more than one volume, however, the parameter list will have its last 12 bytes repeated once for each applicable volume. (The first two bytes of the list indicate the number of volume entries in the list.) If an allocation request is made, word 4 contains the address of the partially completed Job File Control Block (JFCB).

Define security for a specific function by testing for the existence of the appropriate function, establishing the desired level of authorization, and setting the return code in register 15 to indicate either acceptance or rejection.

The use of UDS is oriented toward files. Reference files are identified by fully qualified file names.

UDSEX1 Conventions

The following table summarizes the UDSEX1 linkage conventions.

Feature	Convention												
Attributes	Reentrant, if in a resident area.												
Type	Thread.												
Size	Restricted to the UDS thread region.												
Registers at Entry	<p>Register 1 Address of the parameter list</p> <p>Register 13 Address of an 18-fullword save area</p> <p>Register 14 Return address in the calling module</p> <p>Register 15 Entry address of UDSEX1</p>												
Registers at Return	Registers 2 through 13 must be unchanged. Register 15 must contain a return code.												
Parameters	<p>Word 1 Address of a four-byte field containing the operation requested.</p> <p>Word 2 Address of a 44-byte field containing the file name entered.</p> <p>Word 3 Address of a field with the format:</p> <table border="0" style="margin-left: 40px;"> <tr> <td style="padding-right: 20px;">0</td> <td style="padding-right: 20px;">VOLNUM</td> <td>Halfword indicating the number of 12-byte entries that follow (CODE, VOLUME, SEQ).</td> </tr> <tr> <td>2</td> <td>CODE</td> <td>Four-byte device code for the first volume. This code is the same as that in the UCBTYP field of the UCB.</td> </tr> <tr> <td>6</td> <td>VOLUME</td> <td>Six-byte volume name.</td> </tr> <tr> <td>12</td> <td>SEQ</td> <td>Two-byte sequence number. Note that the fields identified by offsets 2 through 12 may be repeated once per volume.</td> </tr> </table> <p>Word 4 Address of a partially completed JFCB, if the ALLOCATE function was requested; otherwise, binary zeros.</p>	0	VOLNUM	Halfword indicating the number of 12-byte entries that follow (CODE, VOLUME, SEQ).	2	CODE	Four-byte device code for the first volume. This code is the same as that in the UCBTYP field of the UCB.	6	VOLUME	Six-byte volume name.	12	SEQ	Two-byte sequence number. Note that the fields identified by offsets 2 through 12 may be repeated once per volume.
0	VOLNUM	Halfword indicating the number of 12-byte entries that follow (CODE, VOLUME, SEQ).											
2	CODE	Four-byte device code for the first volume. This code is the same as that in the UCBTYP field of the UCB.											
6	VOLUME	Six-byte volume name.											
12	SEQ	Two-byte sequence number. Note that the fields identified by offsets 2 through 12 may be repeated once per volume.											
Return Codes	<p>0 Allow the request.</p> <p>4 Disallow the request.</p>												

Feature	Convention
Considerations	a Is loaded once per call of UDS.
	b Can be link edited, or loaded dynamically.

UDVSX0 - Usage Control of UDS/UDVS VSAM SERVICES

UDVSX0 is a user-written service routine that can be used to control and/or restrict the use of UDS/UDVS VSAM SERVICES.

When Com-plete is initially installed, no UDVSX0 module is available. This routine must be coded with the standard Com-plete exit requirements and must be link edited with the UDS/UDVS load module.

Note:

UDVSX0 is called for each VSAM SERVICES operation.

This chapter covers the following topics:

- How to Use UDVSX0
 - UDVSX0 Conventions
-

How to Use UDVSX0

At the time UDVSX0 is called, register 1 points to the IDCAMS control statement that will be executed. The control statement area has a contiguous length of 280 bytes. It will be presented to IDCAMS as 4 lines with 70 bytes each. A ZERO return code of 0 (zero) in register 15 will allow the operation. A non-zero return code in register 15 disallows the operation.

Note that the exit can modify the control statement area.

UDVSX0 Conventions

The following table summarizes the UDVSX0 linkage conventions.

Feature	Convention	
Attributes	None required.	
Type	Online.	
Size	No restrictions.	
Registers at Entry	Register 1	Points to the control statement area, contiguous, 4 lines at 70 bytes each
	Register 13	Address of the 18-fullword MVS-compatible save area
	Register 14	Return address
	Register 15	Entry point address
Registers at Return	Registers must be unchanged except register 15, which contains the return code.	
Return Codes	0	Allow operation
	non-zero	Disallow operation
Considerations	Must be link edited with UDS.	

UDYEX1 - Control Dynamic Allocation/Deallocation of Datasets using UDYD (MVS only)

UDYEX1 is a user-written exit which can control allocation and deallocation of datasets using the utility UDYD. A default exit is provided on the distributed source and load datasets which only allows control users to allocate and deallocate datasets.

This chapter covers the following topics:

- How to use UDYEX1
 - UDYEX1 Conventions
-

How to use UDYEX1

When called, R1 points to a parameter list which contains the address of three parameters as follows:

- Parm1 One byte code indicating the function to be performed: A: Dataset allocation D: Dataset deallocation
->
- Parm2 44 byte field containing the name of the dataset being allocated or deallocated.
->
- Parm3 6 byte field containing the volser of the dataset being allocated or deallocated if provided by the user. Please note that for a deallocate request, this is not required and therefore may not be filled out.
->

Based on the above information, the exit can decide whether this request should be allowed or not. To allow the request, the user must simply return "0" in RF; to disallow the request return "4" in RF.

UDYEX1 Conventions

The following table summarises the UDYEX1 linkage conventions:

Feature	Convention
Attributes	None required
Type	Online
Registers at entry	R1 points to the parameter list as described above RD points to a 18F savearea which can be used by the exit
Registers at return	All registers must be returned unchanged with the exception of RF which contains the return code.
Return codes	0 Allow request 4 Disallow request
Considerations	Must be link-edited with UDDYN

UEDTB1 - Library Code Table

UEDTB1 is a user-written module that defines the two-character library identification codes used to refer to Com-plete source libraries. The module is loaded by the utility programs UED, UEDIT, and UPDS/USERV, and contains no executable logic.

The user-written table UEDTB1 can be link edited as part of the utility programs themselves, depending upon performance considerations. In the case of UEDIT, UEDTB1 is not link edited into the UEDIT module itself, but is instead link edited as part of the UEBP, UEPDIN, and UEPROF. If the table is not link edited as part of the utilities, the table is loaded at program invocation time. It can also be made resident in Com-plete's storage by specifying sysparm RESIDENTPAGE=UEDTB1.

UEDTB1 is a set of macro statements, each of which defines a specific library and its assigned two-character code. Library security cannot be defined within UEDTB1, but a library code can be restricted to read-only status. If library security is desired, the ULSRSEC routine must be modified, or the appropriate user-written exit routine for the accessing program (e.g., UEDIT) must be written. All Com-plete utility programs that enable libraries to be accessed (e.g., UED, UEDIT, USERV, and UPDS) call the user-written exit ULSRSEC for this purpose. In addition, since these programs also load module UEDTB1, security can be established for entries in UEDTB1 in a generalized fashion without specific knowledge of the contents of the module (UEDTB1).

Dummy UEDTB1 modules are distributed with the Com-plete system as members of the distribution source library and the distribution load library.

This chapter covers the following topics:

- How to Use UEDTB1
- The CMEDTB1 Macro

How to Use UEDTB1

The library identification table UEDTB1 consists of three types of statements:

- TITLE;
- CMEDTB1;
- END.

The TITLE and END statements are standard Assembler language statements and are not discussed here.

The CMEDTB1 Macro

The CMEDTB1 statement is a macro instruction, distributed in the Com-plete distribution source library, with the format:

```

CMEDTB1 ID=xx,DSN=library
      [,VSECAT=ccccccc]
      [,VSELIB=type]
      [,VSESUB=ssssssss]
      [,VOL=dddddd]
      [,OUTPUT=type]
      [,ACM=acm]
      [,USRDIS=YES|NO]
      [,STOW=YES|NO]

```

The arguments are:

ID=xx

Required.

Specifies the two-character library identification code to be assigned to the library being defined.

Here, *xx* is any two alphanumeric characters, the first of which must be alphabetic.

DSN=library

Required.

Specifies the name of the library identified by the two-character library code.

Note that *library* must be the same as the fully qualified cataloged library name. If the library is not cataloged, the optional argument VOL must be specified.

For LIBRARIAN and PANVALET libraries, *library* is the DD name associated with the LIBRARIAN or PANVALET file in the Com-plete initialization procedure.

Note:

LIBRARIAN and PANVALET libraries must be allocated by a job control allocation statement in the Com-plete initialization procedure. The DD name used must be specified by the DSN keyword argument.

For the PC access method, *library* can be used as a comment field.

VSECAT=ccccccc

Is used only for VSE; ignored for others.

Default: VSECAT=IJSYSCT

Specifies the file name for the VSAM Catalog for the specified library. The file name must be from one to seven characters and must be present in either standard-, partition-, or temporary-label before Com-plete startup.

VSELIB=type

Required for VSE; ignored for MVS.

Default: VSELIB=NO

Specifies the type of VSE library to be defined in this entry. Possible values are:

AL -VSAM library

BL -non-VSAM library

Note:

In a VSE environment, the user must define a VSAM (AL) library to be accessed by Com-plete in either UEDTB1 or in an online definition by the UL function of UUTIL. If the library is not defined in UEDTB1 or via ULIB, it cannot be accessed via USERV, UMAP, UEDIT, or UED.

VSESUB=sssssss

Is used only for VSE; ignored for others.

Default: *

Specifies the VSE sublibrary for the specified library. This value is a one- to eight-character sublibrary name.

VOL=dddddd

Optional

Specifies the volume serial number of the disk volume on which the library is to be found.

Default: If VOL is omitted, the system catalog is used to locate the file accessed.

Note:

Use this argument only if the library is not cataloged.

Note for VSE: Do not specify this argument if the library resides in VSAM space.

OUTPUT=type

Optional.

Default: OUTPUT=YES

Specifies whether or not the library being defined is read-only. *type* may be specified as YES or NO.

OUTPUT=YES indicates that the library is to be both input and output (that is, a SAVE operation can be performed).

OUTPUT=NO indicates that the library is to be input only (that is, the library cannot be modified).

ACM=acm

Optional.

Default: ACM=PO

Specifies the type of access method to be used to access the library being defined. Here, *acm* can be specified as PO, PS, PANVALET, LIBRARIAN, or PC.

ACM=PO indicates that the library is a partitioned file and the partitioned file access method is to be used.

ACM=PS indicates that the library is a sequential file and the sequential access method is to be used.

ACM=PANVALET indicates that the library is a PANVALET library and the PAN access method is to be used.

ACM=LIBRARIAN indicates that the library is a LIBRARIAN library and the LIBRARIAN access method is to be used.

ACM=PC indicates that the logical library is contained in a personal computer disk file and the PC access method is to be used.

USRDIS=YES|NO

Optional

Default: USRDIS=YES

USRDIS=NO prevents the display of STOW user data information.

STOW=YES|NO

Optional. (MVS only.)

Default: STOW=YES

STOW=NO causes all STOWs from UED or UEDIT not to write Com-plete's user data.

Note:

The first statement in this assembly must be COPY CCGLOBS to set the correct operating system.

ULHMX1 - Hello Message Exit

ULHMX1 is a user-written routine called by Com-plete's hello message transient, TTOCHM, before the hello message is sent to a terminal.

ULHMX1 not only allows you to modify the contents of the hello message before sending it to a terminal, but it can also indicate that no hello message is to be sent to a specific terminal.

ULHMX1 is entered once for each terminal that is to receive a hello message, either during Com-plete initialization if sysparm HELLOMESSAGE=YES is specified, or during processing of the HMSG operator command. ULHMX1 is link edited with the Com-plete nucleus.

This chapter covers the following topics:

- How to Use ULHMX1
 - ULHMX1 Conventions
-

How to Use ULHMX1

Upon entry to ULHMX1, a set of parameters is provided to supply the exit with the information necessary to perform its function. The first parameter is the address of a message switching control block, as built by MCALL MESGCB, specifying the message class codes and other data for the hello message. The second parameter is the address of the message text to be sent. The third parameter is the address of a halfword containing the length of the message to be sent.

The fourth parameter is the address of the TID number to receive the message. The TID number is in the form of an eight-character, printable number. For example, TID 178 would be "00000178". The fifth parameter is the address of a halfword containing the number of terminals to receive the message. This halfword is always one (1). The sixth parameter is the address of the installation ID and is used to identify the originator of the message. The installation ID is eight characters long.

The addresses in the parameter list can be changed to specify different messages and lengths. Note that the locations pointed to by the parameter list must not be modified.

ULHMX1 Conventions

The following table summarizes the ULHMX1 linkage conventions.

Feature	Convention	
Attributes	None required.	
Type	Nucleus.	
Size	No restriction.	
Registers at Entry	Register 1	Address of the parameter list
	Register 2	Address of the Com-plete COMREG
	Register 13	Address of an 18-word save area
	Register 14	Address of the return point in TTOCHM
	Register 15	Address of the ULHMX1 entry point
Registers at Return	Registers 2 through 13 must be unchanged.	
Parameters	Word 1	Address of the MESGCB
	Word 2	Address of the message text
	Word 3	Address of a halfword message length
	Word 4	Address of the TID
	Word 5	Address of the halfword terminal count (1)
	Word 6	Address of the installation ID
Return Codes	0	Send message.
	4	Do not send message.
Considerations	a	Executes in supervisor state, and key of Com-plete.
	b	Cannot issue Com-plete user functions.

ULMSBTCH - Batch Output User Exit

ULMSBTCH is a user-written routine used to examine and/or modify the parameters which the Com-plete batch spool output routines subsequently use to generate a DYNALLOC/SEGMENT call during batch printout spooling.

ULMSBTCH is dynamically loaded during Com-plete initialization. The module is invoked with service routine status (that is, it is in Com-plete's protect key). This means that if an abend occurs in the exit, all of Com-plete terminates abnormally.

This chapter covers the following topics:

- How to use ULMSBTCH
 - ULMSBTCH conventions
-

How to use ULMSBTCH

On entry to ULMSBTCH, a set of parameters is received in the form of a fullword pointed to by register 1. On return from the exit, the response in register 15 will be examined and processed.

ULMSBTCH conventions

Registers on entry

Register 1: Address of parameter area

Register 13: 18-fullword save area

Registers at return

Register 15: Response code

All other registers must be unchanged.

Parameters

ULSTNAME	DS	CL8	Printout name	(X)
ULSTNUM	DS	AL2	Printout number	
ULSTCOPY	DS	AL2	Printout copies	(X)
ULSTFORM	DS	CL4	Printout form	(X)
ULSTHEAD	DS	XL1	Nr. of header pages to be generated (0=none, 2=default)	(X)
ULSTCLAS	DS	CL1	Output Class	(X)
ULSTOTID	DS	AL2	Original TID number	
ULSTNODE	DS	CL8	Output node	(X)
ULSTUSER	DS	CL8	Userid	(X)
ULSTOUSR	DS	CL8	Originating userid	
ULSTFLG1	DS	XL1	Flag byte	(X)
ULSTMNOS	EQU	X'80'	When on, no separators will be printed (that is, no skip to channel will be issued).	
ULSTHDR1			First header line	(X)
ULSTHDR2			Second header line	(X)
ULSTHDR3			Third header line	(X)

Note that macro CMULST now generates the appropriate DSECT. This must be used to remain compatible with future releases of Com-plete.

(X) indicates that the field can be modified by the user exit.

The header generated by the Batch output facility consists of three blocked lines of output:

```

ULSTOUSR
ULSTNAME
ULSTOUSR

```

These fields may be modified to suit your installation requirements.

Return codes

- 0 Com-plete issues the DYNALLOC/SEGMENT using the parameters specified.
- 4 Com-plete assumes that the user exit has issued a valid DYNALLOC/SEGMENT call. The user exit is responsible for issuing and validating the DYNALLOC/SEGMENT call.
- 8 Com-plete rejects the printout request.

Using AFP printers

You can use ULMSBTCH to setup an output descriptor for, e.g., AFP printers and tell Com-plete to use this output descriptor at the DYNALLOC macro for SYSOUT. To achieve this, proceed as follows:

1. Create the output descriptor using the IBM macro OUTADD. Specify PAGEDEF and FORMDEF based on the contents of ULSTFORM using your own relation table and/or algorithm.
2. Put the name of the output descriptor into ULSTNAME and switch on the x'40' bit (ULSTMOUT) in ULSTFLG1 to indicate there is an output descriptor.
3. ULMSBTCH itself is responsible for maintaining information about the output descriptors it already added, issuing OUTDEL macros when/if necessary, etc.

ULMSDISK - Dynamic Printer Allocation User Exit

ULMSDISK is a user-written routine called by the Com-plete printout spooling system immediately before and after allocation of terminal information control block (TIB) for a dynamically acquired printer device. Possible uses of this exits are:

- Test for valid printer name;
- Modification of the TIB entry for this printer. Specifically, this enables you to overwrite the TIBNAME field and so re-route the printout onto an alternative printer. This is useful when there is no rigid relationship between logical printer definitions (for example, in NATURAL) and the physical printer itself (that is, VTAM LU definition).

This chapter covers the following topics:

- How to Use ULMSDISK
 - ULMSDISK Conventions:
-

How to Use ULMSDISK

Upon entry to ULMSDISK the following registers are set:

Register 5 contains the address of the filled TIB entry, or 0 if the call to ULMSDISK is before dynamic allocation. If register 5 is 0, Register 1 contains the address of the printer name.

ULMSDISK Conventions:

The following table summarizes the ULMSDISK linkage conventions.

Feature	Convention
Attributes	Reentrant.
Type	Nucleus.
Registers at Entry	<p>Register 1 Address of the printer name if Register 5 is 0.</p> <p>Register 5 Address of the TIB entry, or 0 if the call is before dynamic allocation.</p> <p>Register 13 Address of an 18-fullword savearea</p> <p>Register 14 Return address.</p> <p>Register 15 Entry address.</p>
Registers at Return	<p>Before allocation:</p> <p>Register 15 Returncode; all others must remain unchanged.</p> <p>After allocation: all registers must remain unchanged.</p>
Parameters	None.
Return Codes	<p>Before allocation:</p> <p>0 Dynamic allocation allowed.</p> <p>4 Dynamic allocation disallowed.</p> <p>After allocation: none</p>
Considerations	<p>a Executes in supervisor state and Com-plete key.</p> <p>b Cannot issue COM-LETE user functions.</p> <p>c Called in Com-plete's key.</p> <p>d Active thread will generally be in a different key to Com-plete's key.</p>

ULINUSER - Com-plete Initialization Exit

ULINUSER is a user-written routine that is loaded during Com-plete initialization and termination processing.

This chapter covers the following topics:

- How to Use ULINUSER
- ULINUSER Conventions

How to Use ULINUSER

The ULINUSER routine can execute any SVC except the Com-plete SVC or the MCALL SVC. The routine is given the address of Com-plete's communications region in register 2, and the address of an 18-word register-save area in register 13. Note that registers must be saved and restored. The routine is given control in problem state in Com-plete's protect key and is run in the main task.

ULINUSER Conventions

The following table summarizes the ULINUSER linkage conventions.

Feature	Convention
Attributes	Need not be reentrant or reusable.
Type	Nucleus.
Size	No size limitation.
Registers at Entry	<p>Register 1 at initialization call: points to a 4-byte constant 'INIT' at termination call: points to a 4-byte constant 'TERM'</p> <p>Register 2 Points to Com-plete's communications region (see the COMREG member in the distributed source library).</p> <p>Register 13 Points to an 18-word save area.</p>
Registers at Return	All registers must be saved and restored.
Parameters	None.
Return Codes	None.
Considerations	Executes in supervisor state, key of Com-plete.

ULOGX1 - ULOG Security Exit

ULOGX1 is a user-written routine called by the Com-plete ULOG utility program prior to various events, which are listed below.

When a terminal user logs onto Com-plete via the ULOG ON request, a ULOG Info Table (CCUIT) is created for the user before passing control to ULOGX1. This info table contains information taken from the user ID accounting block for the user ID, the data entered at the terminal, and miscellaneous information. For more information on the user ID accounting block, see the chapter Accounting.

ULOGX1 allows the installation to modify the contents of the CCUIT block before the construction of the user ID accounting block. It also enables secondary levels of user-defined security checking.

This chapter covers the following topics:

- How to Use ULOGX1
 - Compatibility with ULGEX1
 - ULOGX1 Conventions
-

How to Use ULOGX1

Upon entry to ULOGX1, register 1 points to a parameter list. The first word points to the Com-plete ULOG Info Table (CCUIT), whose DSECT is listed in ULOG Info Table. The halfword UITCODE contains the function code, which corresponds to the ULOG message 'ULGnnnn', where nnnn = function code.

The second word, where applicable, points to the message(s) returned by the external security system.

The third word points to a halfword containing the number of 80-character messages in the area pointed to by word two.

The exit is activated in Com-plete's key.

Compatibility with ULGEX1

Previous versions of Com-plete delivered an example ULOGX1 module serving as an interface to the old ULGEX1 exit routine. From version 4.5 onwards, Com-plete no longer supports this "link". If you still use an old ULGEX1 with Com-plete version 4.4, you must rewrite ULOGX1 to take over responsibility of your ULGEX1.

ULOGX1 Conventions

The following table summarizes the ULOGX1 linkage conventions.

Feature	Convention	
Attributes	None required.	
Type	No restrictions.	
Size	No restrictions.	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address within ULOG
	Register 15	Entry address within ULOGX1
Registers at Return	Registers 12 through 14 must be unmodified.	
Parameters	Word 1	Address of the ULOG Info Table
	Word 2	Address of External Security message(s), zero if none.
	Word 3	Address of a halfword containing the number of External Security message(s).
Return Codes	0	Allow the function.
	4	Disallow the function.
Considerations	a	Called by ULOG.
	b	Linkage is dynamic, with COLOAD.

A description of some important UITCODEs and the function ULOGX1 may perform at the respective calls is given below:

Code	Function to be performed by ULOGX1
0003	Logon processing is complete. The 'ULG0003 logon successful...' message can be suppressed by setting UITWRM=N, UITNBROD=N inhibits ULOG showing broadcast information to the user.
0004	This is your last chance to modify UIT fields before the UAB is built.
0005	Logoff processing starts, time to perform site-specific user disconnection.
0006	This user is already logged on. If you wish to allow multiple sessions with the same user ID, set UITALLW to C'Y'. UITADTIB contains the address of the TIB of the user's other session.
0008	When required, modify UITUID now before user ID and User profile definition records are retrieved from the System Data Container
0013	External security system messages are now ready to be processed. ULOGX1 may
-""-	<ul style="list-style-type: none"> a set UITSAFIM=N to suppress message display at all b modify / edit the message(s) c change the number of messages to be displayed by modifying the halfword third parameter passed to the exit.
0038	Modify UITCMOD now when either no model name is available so far but one is required, or the model currently set does not suit your needs.

ULOPADAB - Adabas User Exit

ULOPADAB is a user-written routine called by Com-plete's Adabas interface routine, TLOPADAB, before processing an Adabas command from a Com-plete application program.

ULOPADAB, which allows the installation to examine and/or modify the contents of the Adabas parameters, is entered once for each Adabas command issued by a Com-plete program.

Note:

Com-plete user program functions are not available to ULOPADAB.

This chapter covers the following topics:

- How to Use ULOPADAB
 - ULOPADAB Conventions
-

How to Use ULOPADAB

Upon entry to ULOPADAB, the Adabas parameter list is provided. Each parameter is documented in the **Adabas Operations** documentation.

The data provided by the parameter list can be examined and/or altered, as required, by ULOPADAB.

ULOPADAB Conventions

The following table summarizes the ULOPADAB linkage conventions.

Feature	Convention																		
Attributes	Reentrant.																		
Type	Nucleus.																		
Size	No restriction.																		
Registers at Entry	<table> <tr> <td>Register 1</td> <td>Zero</td> </tr> <tr> <td>Register 2</td> <td>Address of the Com-plete COMREG</td> </tr> <tr> <td>Register 3</td> <td>Address of the Com-plete UPCB for the caller</td> </tr> <tr> <td>Register 4</td> <td>Address of the Com-plete THCB for the caller</td> </tr> <tr> <td>Register 5</td> <td>Address of the Com-plete TIB for the caller</td> </tr> <tr> <td>Register 6</td> <td>Address of the caller's Adabas parameter list</td> </tr> <tr> <td>Register 13</td> <td>Address of an 18-word save area</td> </tr> <tr> <td>Register 14</td> <td>Address of the return point in TLOPADAB</td> </tr> <tr> <td>Register 15</td> <td>Address of the ULOPADAB entry point</td> </tr> </table>	Register 1	Zero	Register 2	Address of the Com-plete COMREG	Register 3	Address of the Com-plete UPCB for the caller	Register 4	Address of the Com-plete THCB for the caller	Register 5	Address of the Com-plete TIB for the caller	Register 6	Address of the caller's Adabas parameter list	Register 13	Address of an 18-word save area	Register 14	Address of the return point in TLOPADAB	Register 15	Address of the ULOPADAB entry point
Register 1	Zero																		
Register 2	Address of the Com-plete COMREG																		
Register 3	Address of the Com-plete UPCB for the caller																		
Register 4	Address of the Com-plete THCB for the caller																		
Register 5	Address of the Com-plete TIB for the caller																		
Register 6	Address of the caller's Adabas parameter list																		
Register 13	Address of an 18-word save area																		
Register 14	Address of the return point in TLOPADAB																		
Register 15	Address of the ULOPADAB entry point																		
Registers at Return	All registers must be returned unchanged.																		
Parameters	Refer to the Adabas Operations documentation.																		
Return Codes	None.																		
Considerations	<table> <tr> <td>a</td> <td>Executes in supervisor state, and key of the user (thread).</td> </tr> <tr> <td>b</td> <td>Cannot issue the Com-plete user function (MCALL).</td> </tr> <tr> <td>c</td> <td>Called in the protect key in which the user is running.</td> </tr> <tr> <td>d</td> <td>Adabas control block (ACB) and user block (UB) will be in the key of the user.</td> </tr> <tr> <td>e</td> <td>Other areas pointed to by the Adabas parameter list may</td> </tr> </table>	a	Executes in supervisor state, and key of the user (thread).	b	Cannot issue the Com-plete user function (MCALL).	c	Called in the protect key in which the user is running.	d	Adabas control block (ACB) and user block (UB) will be in the key of the user.	e	Other areas pointed to by the Adabas parameter list may								
a	Executes in supervisor state, and key of the user (thread).																		
b	Cannot issue the Com-plete user function (MCALL).																		
c	Called in the protect key in which the user is running.																		
d	Adabas control block (ACB) and user block (UB) will be in the key of the user.																		
e	Other areas pointed to by the Adabas parameter list may																		

ULSRMPEX - Modify PF Key Codes

You can provide user exit ULSRMPEX to make the required modifications. The exit will be loaded during Com-plete initialization and be called with the following register conventions:

R1 address of parmlist (explained below)

R2 COMREG

R5 TIB

RD standard save area

RE return addr

RF entry point

All registers have to be restored before return.

The program will be called according to its RMODE specification and has to return via B COMRETRN (in COMREG). Since it will be executed under control of thread-task it has to be reentrant and can't use any Com-plete OP-Codes (MCALL).

The parmlist contains 4 parms:

P1 address of a halfword function code x'0010' called after AID/MRCBECDE translation

P2 address of mapname (CL6)

P3 and P4 are dependent on the function code (P1).

For X'0010' they have the following meaning:

P3 address of the AID in the input buffer (CL1)

P4 address of the enter code (MRCBECDE) in the MRCB (CL2)

The exit can set the MRCBECDE according to your needs. It should be noted that all Com-plete utilities and the related maps (prefix = C'U') are dependent on the standard MRCBECDE values of Com-plete and will not work properly if any translation is performed.

ULSRPSFS - User-Written Service Routine

ULSRPSFS is a user-written service routine that can be used to authorize the use of specific application functions, programs, modules, or Com-plete utility programs.

Note:

Because security afforded by this service routine is system-wide, thorough design and analysis is recommended before implementation in order to ensure minimal impact on performance.

When Com-plete is initially installed, a dummy ULSRPSFS module exists as a member of the distribution source library to serve as a guide to the user. This routine must be coded and assembled and can either be link edited as a routine in the Com-plete control program, or loaded dynamically during initialization. The initial Com-plete control program contains a dummy ULSRPSFS service routine that performs no security checking.

Since ULSRPSFS executes as a service routine of Com-plete, it will be dispatched in supervisor state using the key assigned to Com-plete. Consequently, if an abnormal termination occurs while ULSRPSFS is executing, Com-plete also abends.

Note that because ULSRPSFS is entered frequently, it is not advisable for it to perform any I/O functions. Tables and any referenced subroutines should be made part of the service routine.

This chapter covers the following topics:

- How to Use ULSRPSFS
 - ULSRPSFS Conventions
-

How to Use ULSRPSFS

At entry to ULSRPSFS, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of the return code indicating the security status. Word 2 contains the address of the user ID accounting block. This control block is created for each user at successful logon to Com-plete. If a user ID requests a function protected by ULSRPSFS, a call is made to ULSRPSFS.

Words 3 and 4 of the parameter list identify the function requested by a program, and word 6 identifies the program making the request. Since authorization for access to most functions is restricted on a user ID basis rather than on a program basis, it is necessary to also identify the terminal and/or batch user making the request. This is accomplished by passing the required information through words 2 and 5 of the parameter list. Word 7 points to two status bytes; the first indicates whether or not a program has been attached, and the second indicates whether or not an Adabas call exists at the time of entry to the ULSRPSFS.

Security violations are indicated in ULSRPSFS by setting a return code. The physical location of the return code is in the routine that passes control to ULSRPSFS. The address of the return code is passed as word 1 of the parameter list. If the function requested is not to be allowed, the return code must be set to 4; otherwise, the return code must be set to 0. At entry to ULSRPSFS, the return code is always initialized to 0.

ULSRPSFS Conventions

The following summarizes the ULSRPSFS linkage conventions.

Attributes

Reentrant.

Type

Nucleus.

Size

No restrictions.

Registers at Entry

Register 1	Address of the parameter list
Register 13	Address of an 18-fullword save area
Register 14	Return address in the calling module
Register 15	Entry address within ULSRPSFS

Registers at Return

Registers 2 through 13 must be unchanged.

Parameters

Word 1	Address of a return code halfword
Word 2	Address of the user ID accounting block
Word 3	Address of a request-type halfword:

	<p>0 Program call (terminal).</p> <p>4 LOAD request.</p> <p>8 LOADT request.</p> <p>12 FETCH request.</p> <p>16 SDOPEN request.</p> <p>32 ATTACH request, where R3 points to the UPCB. R3+12 contains the address of a parameter list: Parm 1 is the address of the eight-character program name of the program being attached. Parm 2 (optional) is the address of the data area being passed. Parm 3 (optional) is the length of the data area being passed.</p> <p>36 LINK, LOAD, XCTL, COLINK, COLOAD, COXCTL request, where R3 points to the UPCB. R3+8 is R0 at the time of request. R3+12 is R1 at the time of request. R3+68 is RF at the time of request. R3+12 is the address of the user parameter list. The first parameter in the parameter list points to an eight-character program name. For LINK, LOAD, and XCTL; R0, R1, and RF are as expanded by MVS LINK, LOAD, and XCTL.</p> <p>40 Indicates that word 8 points to an eight-byte location containing the error message ID in character format (e.g., ZSR0006).</p> <p>44 Catch-all code; indicates that the parameter list is extended by three additional fullwords (words 9, 10, and 11) used to identify the operation code or SVC being requested that does not fall within the range of other operation codes. Com-plete OP codes are found in member CCTOP in COM.SOURCE. These records are only written if APPLYMOD 3 is specified.</p> <p>100 Logoff.</p>
Word 4	Address of the requested eight-character name padded with blanks. This is the name of the program being loaded, the name of the SD file being opened, etc.
Word 5	Address of a halfword containing the TID. If a batch program, the high-order byte of the halfword is an X'FF'.

Word 6	Address of the requesting program name (job name for batch, except request types 0 and 100).					
Word 7	Address of the two status bytes:					
	<table> <tr> <td>Byte 1</td> <td>A = Attached program</td> <td>sp = Non-attached program</td> </tr> <tr> <td>Byte 2</td> <td>A = Open to Adabas at entry</td> <td>sp = Not open to Adabas</td> </tr> </table>	Byte 1	A = Attached program	sp = Non-attached program	Byte 2	A = Open to Adabas at entry
Byte 1	A = Attached program	sp = Non-attached program				
Byte 2	A = Open to Adabas at entry	sp = Not open to Adabas				
Word 8	Not used.					
Word 9	Address of a one-byte area:					
	X'FF' SVC issued					
	X'00' Com-plete OP issued					
	This entry exists only if word 3 points to a halfword 44 and only if OP trapping is enabled.					
Word A	Address of the halfword SVC or OP. This entry exists only if word 3 points to a halfword 44.					
Word B	Address of the UPCB. This entry exists only if word 3 points to a halfword 44.					

Return Codes

0	Allow the request.
4	Disallow the request.

Considerations

a	Runs as a portion of the Com-plete nucleus; it is there fore advisable to maintain a backup either of this module or of the entire Com-plete control program.
b	Tables and reference routines must be made part of the service routine.
c	I/O must be avoided.
d	MCALL is not permitted.
e	Com-plete functions are not permitted.
f	Overlays cannot be used.
g	GETMAINS must be used for storage.
h	GETMAINS are from the Com-plete region, not thread.
i	Called in Com-plete's key.
j	All control blocks and areas passed for the use of the exit will be in Com-plete's key.
k	Active thread will generally be in a different key to Com-plete's key.

ULSRRJE - Remote Job Entry User Exit

ULSRRJE is a user-written routine used to examine and/or modify the RJE input data submitted for background processing via the RJE function call. This exit is similar in capability and function to the UEDIT RJE exit UUEEX3. ULSRRJE, however, has the advantage of being able to process RJE requests from any source, not just the editor. Possible uses for this exit include:

- To syntax-check JCL;
- To enforce installation standards;
- To insert UQ security statements into the input stream.

ULSRRJE is loaded dynamically at Com-plete initialization. The address of the module is found in COMREG, field AUSRRJE. It is given control via call from TLSRRJE before RJE begins: once before each card image is passed to the operating system, and at the end of the data.

The module is invoked with service-routine status (that is, it is in Com-plete's protect key, and all Com-plete and supervisor calls are valid). Note that if an abend occurs in the exit, Com-plete terminates abnormally. The exit routine must be reentrant to allow for multiple thread access.

Com-plete user program functions (MCALL) are not available to the exit. Any environmental data (user ID, TID, etc.) must be obtained from the standard Com-plete control blocks passed to the exit.

This chapter covers the following topics:

- How to Use ULSRRJE
 - ULSRRJE Conventions
-

How to Use ULSRRJE

Upon entry to ULSRRJE, registers are set according to the Com-plete convention (R2=COMREG, R3=UPCB, R4=THCB, R5=TIB, R13=18-fullword save area, R14=return address, R15=entry point).

R1 contains the address of a parameter list. Word 1 of the parameter list contains the address of a return code halfword that has been initialized to zero. Word 2 of the parameter list contains the address of the requesting RJE parameter list. Word 3 of the parameter list contains the address of the call-type flag byte (character). Word 4 of the parameter list contains the address of the statement being submitted.

Note that the last parameter address is not valid if the call-type byte is S.

Upon return from the exit, the return code halfword is examined and processed.

At the end of the job stream, the call-type is set to E and ULSRRJE is called repeatedly until a return code of 0 or 12 is given.

A statement can be modified by ULSRRJE by simply changing it in the data area addressed by the fourth word and returning with return code 0.

Once a return code of 12 or more is returned by ULSRRJE, no further calls are made to the exit.

ULSRRJE Conventions

The following table summarizes the linkage conventions of ULSRRJE.

Feature	Convention														
Attributes	None required.														
Type	Nucleus.														
Size	No restrictions.														
Registers at Entry	<table> <tr> <td>Register 1</td> <td>Address of the parameter list</td> </tr> <tr> <td>Register 2</td> <td>Address of the COMREG</td> </tr> <tr> <td>Register 3</td> <td>Address of the UPCB</td> </tr> <tr> <td>Register 4</td> <td>Address of the THCB</td> </tr> <tr> <td>Register 5</td> <td>Address of the TIB</td> </tr> <tr> <td>Register 13</td> <td>18-fullword save area</td> </tr> </table>	Register 1	Address of the parameter list	Register 2	Address of the COMREG	Register 3	Address of the UPCB	Register 4	Address of the THCB	Register 5	Address of the TIB	Register 13	18-fullword save area		
Register 1	Address of the parameter list														
Register 2	Address of the COMREG														
Register 3	Address of the UPCB														
Register 4	Address of the THCB														
Register 5	Address of the TIB														
Register 13	18-fullword save area														
Registers at Return	All registers must be unchanged.														
Parameters	<table> <tr> <td>Word 1</td> <td>Address of a return code halfword</td> </tr> <tr> <td>Word 2</td> <td>Address of the requesting RJE parameter list</td> </tr> <tr> <td>Word 3</td> <td>Address of a flag byte indicating the type of call: <table> <tr> <td>S</td> <td>start of the job submission before the first statement</td> </tr> <tr> <td>sp</td> <td>(space) statement being submitted</td> </tr> <tr> <td>E</td> <td>end of the submission following the last statement</td> </tr> </table> </td> </tr> <tr> <td>Word 4</td> <td>Address of the statement being submitted</td> </tr> </table>	Word 1	Address of a return code halfword	Word 2	Address of the requesting RJE parameter list	Word 3	Address of a flag byte indicating the type of call: <table> <tr> <td>S</td> <td>start of the job submission before the first statement</td> </tr> <tr> <td>sp</td> <td>(space) statement being submitted</td> </tr> <tr> <td>E</td> <td>end of the submission following the last statement</td> </tr> </table>	S	start of the job submission before the first statement	sp	(space) statement being submitted	E	end of the submission following the last statement	Word 4	Address of the statement being submitted
Word 1	Address of a return code halfword														
Word 2	Address of the requesting RJE parameter list														
Word 3	Address of a flag byte indicating the type of call: <table> <tr> <td>S</td> <td>start of the job submission before the first statement</td> </tr> <tr> <td>sp</td> <td>(space) statement being submitted</td> </tr> <tr> <td>E</td> <td>end of the submission following the last statement</td> </tr> </table>	S	start of the job submission before the first statement	sp	(space) statement being submitted	E	end of the submission following the last statement								
S	start of the job submission before the first statement														
sp	(space) statement being submitted														
E	end of the submission following the last statement														
Word 4	Address of the statement being submitted														

Feature	Convention
Return Codes	<p>0 Normal return. Process the next statement unless end-of-file.</p> <p>4 Delete the current statement. Valid only if the call-type=space.</p> <p>8 Insert statement before the current card image. Statement to be inserted was placed in the data area pointed to by word 4 of the parameter list. After inserting the statement, TLSRRJE will again call ULSRRJE with the original card image until a different return code is received. A return code of 8 is valid only for call-types of space and E.</p> <p>12 Abort the submission, passing return code 12 to the application program.</p> <p>16 Abort the submission, terminating the application program. An online dump is taken.</p>
Considerations	<p>a Is given control when called from TLSRRJE before RJE begins : once before each card image is sent to the MVS, and at end-of-data.</p> <p>b Must be reentrant in order to allow multiple thread access.</p> <p>c Is invoked with service-routine status in Com-plete's protect key. All supervision calls are valid.</p> <p>d Com-plete user program functions are not available to the exit.</p> <p>e Environment data must be obtained from the standard Com-plete control blocks passed to the exit.</p> <p>f Called in Com-plete's key.</p> <p>g All control blocks and areas passed for the use of the exit will be in Com-plete's key.</p> <p>h Active thread will generally be in a different key to Com-plete's key.</p>

ULSRSEC - User-Written Service Routine

Note:

Because security afforded by this service routine is system-wide in scope, thorough design and analysis are recommended before implementation in order to ensure minimal impact upon performance.

This module is to be used as a single routine to service security check requests. It will eventually replace ULSRPSFS as the user security exit for all Com-plete functions and is intended to replace file security functions currently available in utility program user exits. The utility program user exit interfaces are still supported, but do not need to be related to file security.

This section is to be used as a guide to modifying the ULSRSEC member in the distribution source library. It also refers to the security control area that is described in detail in the distribution source member CCSCA.

This chapter covers the following topics:

- Initialization Overview
 - Initialization Processing
 - How to Use ULSRSEC
 - ULSRSEC Conventions
-

Initialization Overview

When Com-plete is initially installed, a dummy ULSRSEC module exists in the distribution source library to serve as a guide. This routine must be coded, assembled, and link edited. The initial Com-plete control program contains a ULSRSEC routine that enforces standard MVS password protection for the UPDS, UED, UMAP, UEDIT, UDS, and ULIB utility programs.

Since ULSRSEC executes as a service routine of Com-plete, it is dispatched in supervisor state and in the key of Com-plete. Consequently, if an abend occurs while ULSRSEC is executing, Com-plete terminates abnormally.

Because ULSRSEC is entered frequently, any required tables should be loaded during initialization. The work area passed to ULSRSEC can be expanded if more space is needed.

Initialization Processing

ULSRSEC is given control once during Com-plete initialization, so this portion of ULSRSEC need not be reentrant. If a large routine is required, it could be coded in a routine called by ULSRSEC.

Suggested uses for the initialization routine are:

- To set the value of the CSEWORK field in the Com-plete communication region. This HW field contains a value from 0 to 8000; this is the amount of work space to be obtained by calling routines as a work area for this module. Refer to the label DWORK in the ULSRSEC example.

- To load security tables.
- To set up interfaces with security packages.*

* This applies only to security packages which do not support the SAF interface. Com-plete provides data set protection via this facility as standard. See the chapter Software Interfaces for more information.

The register conventions for the initialization functions are as documented for security checks, and the SCAFUNC field is set to INIT.

A mainline routine is provided to process security check requests. Note that this routine must be reentrant. A save area is provided and already pointed to by register 13. Work areas can be appended to the SCA, but the CSEWORK field must have been set to the proper value. Refer to label DWORK in the ULSRSEC example. Note also that any MVS/VSE SVC can be issued, but no MCALL functions are allowed.

How to Use ULSRSEC

Refer to member CCSCA in the distributed source library for the format and field names of the SCA referred to in this section. The SCA is used to pass the description of user requests to ULSRSEC from the Com-plete nucleus and utility programs. The SCAFUNC field describes the type of request. The SCARDESC field describes the type of resource, and the SCARRES field actually names the resource (DSN, PGMNAME, etc.). Note that for the current version of Com-plete, only the values DSN, ULIB, and USR are passed to ULSRSEC in the SCARDESC field.

By examining the SCA, access can be limited to any described resource by setting a return code before returning control to Com-plete.

The following tables summarize the SCA field values upon entry to ULSRSEC by utility or file I/O operation. Note that the SCAFUNC value INIT is not documented in the tables, because it does not relate to any particular utility.

SCA Field	Value	UPDS Utility Command
SCARDESC	'DSN'	-
SCARDSN	File Name	-
SCARVOL	Volume Serial Number	-
SCAFUNC	'INQ'	LIST
SCAFUNC	'READ'	DISPLAY
SCAFUNC	'READ'	PRINT
SCAFUNC	'READ'	COPY
SCAFUNC	'WRIT'	SCRATCH
SCAFUNC	'WRIT'	RENAME
SCAFUNC	'WRIT'	ALIAS
SCAFUNC	'WRIT'	ZAP

SCA Field	Value	UPDS Utility Command
SCARDESC	'DSN'	-
SCARDSN	File Name	-
SCARVOL	Volser	-
SCAFUNC	'INQ'	DISPLAY
SCAFUNC	'INQ'	FIND
SCAFUNC	'INQ'	LIST
SCAFUNC	'CATL'	RECATLOG
SCAFUNC	'CATL'	CATALOG
SCAFUNC	'CATL'	UNCATALOG
SCAFUNC	'CATL'	RENAME
SCAFUNC	'CATL'	BLDA
SCAFUNC	'CATL'	BLDG
SCAFUNC	'CATL'	BLDX
SCAFUNC	'CATL'	DLTA
SCAFUNC	'CATL'	DLTX
SCAFUNC	'CATL'	CONNECT
SCAFUNC	'CATL'	RELEASE
SCAFUNC	'ALLO'	PURGE
SCAFUNC	'ALLO'	SCRATCH
SCAFUNC	'ALLO'	ALLOCATE

SCA Field	Value	UMAP Utility Command
SCARDESC	'DSN'	-
SCARDSN	File Name	-
SCARVOL	Volser	-
SCAFUNC	'WRIT'	Any effort to SAVE the map in the load library
SCAFUNC	'READ'	All operations on temporary maps in the SDfile

SCA Field	Value	UEDIT/UED Utility Command
SCARDESC	'DSN'	-
SCARDSN	File Name	-
SCARVOL	Volser	-
SCAFUNC	'WRIT'	Any SAVE, RSAVE, or FILE that writes to the library
SCAFUNC	'READ'	Any operation other than SAVE, RSAVE, and FILE that does not write to the library

SCA Field	Value	ULIB Utility Command
SCARDESC	'ULIB'	-
SCARDSN	-	-
SCARVOL	-	-
SCARULTP	'PGM'	-
SCAFUNC	'WRIT'	CAT
SCAFUNC	'WRIT'	DEL
SCAFUNC	'INQ'	DISPLAY

SCA Field	Value	User VSAM/BDAM/ISAM Requests Utility Command
SCARDESC	'DSN'	-
SCARDSN	File Name	-
SCAFUNC	'READ'	READ
SCAFUNC	'WRIT'	WRITE
SCAFUNC	'WRIT'	UPDATE

ULSRSEC Conventions

The following table summarizes the ULSRSEC linkage conventions.

Feature	Convention
Attributes	Reentrant.
Type	Nucleus.
Size	No restriction.

Feature	Convention	
Registers at Entry	Register 0	Available
	Register 1	Available
	Register 2	Com-plete Communications Region (COMREG)
	Register 3	User Program Control Block (UPCB)*
	Register 4	Thread Control Block (THCB)
	Register 5	Terminal Information Block (TIB)*
	Register 6	Reserved for future expansion
	Register 7	Available
	Register 8	Available
	Register 9	Available
	Register A	Available
	Register B	Base 2
	Register C	Base 1
	Register D	Save area and SCA address
	Register E	Available
	Register F	Available
Registers at Return	Registers must be restored, except register 15, which must contain a return code.	
Return Codes	0	Security passed.
	4	Security check failed due to lack of or incorrect information. The SCARC field must be set to the following values :
	0	password failed
	4	unexpired file
	8 to 16	reserved
	8	Security failed.
	12	SCA invalid.

Feature	Convention
Considerations	a Referenced tables must be loaded during initialization.
	b Registers must be saved and restored.
	c The module must be reentrant, except for the initialization routine.
	d The module runs as a portion of the Com-plete nucleus; it is therefore advisable to maintain a backup copy of this module or a backup of the entire Com-plete control program.
	e Called in Com-plete's key
	f All control blocks and areas passed for the use of the exit will be in Com-plete's key.
	g Active thread will generally be in a different key to Com-plete's key.

* Not for the INIT call.

UMSEX1 - UM Security Exit

UMSEX1 is a user-written routine called by UM:

- during UM initialization;
- after input from the main menu;
- before actually scheduling a message switching request.

UMSEX1 can be used to impose security restrictions on the use of certain UM functions or destinations.

The UMSEX1 module is loaded during UM initialization; therefore, internal switches can be set and referenced later on.

A dummy UMSEX1 module is distributed with the Com-plete system as a member of the source and load libraries.

Note:

No security exists for UM functions unless established by you.

This chapter covers the following topics:

- How to use UMSEX1
 - UMSEX1 Conventions
-

How to use UMSEX1

Upon entry to UMSEX1, a set of parameters is received in the form of 5 fullword addresses. For the meaning of the words see the parameters in the table below.

Upon return from UMSEX1 to UM, the return code area is checked. If the return code is not zero, the requested operation is aborted and an error message is displayed on the main menu screen.

UMSEX1 Conventions

The following table summarizes the UMSEX1 linkage conventions:

Feature	Convention	
Attributes	Reentrant if in resident area	
Type	Thread	
Siz	Restricted to the UM thread region size	
Registers at entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address of UMSEX1
Registers at return	All registers except R15 must be unchanged.	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of a four byte function code
	Word 3	Address of the UM00 map area
	Word 4	Address of the destination list (100 entries, 10 bytes per entry)
	Word 5	Address of a halfword showing the number of already selected destinations
Return Codes	0	Allow the request
	4	Disallow the request

Function codes passed via word 2:

F'0' Initialization

F'4' After main menu input

F'8' About to schedule the message

Data area passed via word 3:

UM00D	DSECT	LAYOUT OF MAP UM00
UM00FUNC	DS	CL2FUNCTION CODE (SM,SG,...)
UM00MSGs	DS CL132	MESSAGE TEXT
UM00MSGM	DS CL1	'MORE TEXT' INDICATOR
UM00DEST	DS 7CL8	7 DESTINATIONS
UM00CLAS	DS CL20	CLASS CODES
UM00DL	EQU*-UM00D	LENGTH OF THIS DSECT

USTKX1 - USTACK User Exit

USTKX1 is a user-written routine called before writing the screen and after reading the screen. This means that you can use this exit to perform their own screen I/Os.

In the exit, you can modify the screen buffer and thus build your own map for USTACK. If the exit is called after read processing, the input can be analyzed by the exit; if the exit returns a return code 4, the screen buffer is rebuilt. A DSECT which describes the screen buffer is given to you in the CMUSTK macro.

Because of the heavy use made of USTACK in a running Com-plete system, you must take great care when writing this exit.

This chapter covers the following topics:

- Using USTKX1
 - USTKX1 Conventions
-

Using USTKX1

When calling the exit before write processing:

The write operation is indicated by a value of 0 in register 0. Register 1 points to the screen buffer described in macro CMUSTK. The exit-call register 15 is then checked for the return code. A return code of 0 means normal write/read processing continues. A return code of 4 means read/write processing is performed by the user exit. The exit is called again with the value of 4 in register 0 to indicate read operation.

When calling the exit after read/write processing:

The read operation is indicated by a value of 4 in register 0. Register 1 points to the screen buffer. The exit-call register 15 is then checked for the return code. A return code of 0 means normal analyzing of the input is performed by USTACK. A return code of 4 means input analysis is skipped, and the screen buffer is rebuilt.

USTKX1 Conventions

Feature	Convention	
Attributes Size	none required.	
Registers at entry	Register 0 Register 1 Register 13 Register 14 Register 15	indicates operation: 0 write operation 4 read operation points to screen buffer address of an 18-fullword save area return address in the calling module entry address within USTKX1
Registers at return	Register 15 Register 2 thru' 13	must have the return code must be unchanged.
Return Codes	Register 15	
	0 4	normal processing, read/write processing, input analysis performed by USTACK read/write processing performed by user exit, input analysis performed by user exit.

USTRE1 - USTOR User Exit

USTRE1 is a user-written routine called by the USTOR utility program before the execution of any command function requested by the terminal operator. This routine allows you to define security restrictions on the use of the various functions.

Each time the terminal operator issues a new command or presses the ENTER key, a call is made to USTRE1 before the requested function is serviced. Consequently, the installation may restrict, permit, or eliminate any or all of the USTOR functions.

USTRE1 can issue any Com-plete function in the process of determining the security restrictions to be imposed upon the terminal user. Sample functions include:

- COLOAD Loads a table containing security information.
- COLINK Invokes another user-written program.
- GETCHR Obtains information about the terminal user.
- WRTC Requests additional information.
- READ Obtains the requested information.
- MESGSW Logs a security violation.

Because the USTRE1 module is only loaded once per invocation of USTOR, internal switches can be set and subsequently referred to. Note that each new invocation of USTOR loads a new version of USTRE1, thus causing the switches to be reset.

A dummy USTRE1 is distributed with the Com-plete system as a member of the distribution source library and the distribution load library.

Note:

No additional security exist for USTOR functions unless it is established by you.

This chapter covers the following topics:

- How to Create USTRE1
 - How to Use USTRE1
 - USTRE1 Conventions
-

How to Create USTRE1

To create a new USTRE1 exit routine, proceed as follows:

Step 1

Code the desired module using the member USTRE1 in the distribution source library as a guide.

Step 2

Assemble and link the new module into the user load library.

Step 3

Delete the old USTRE1 from the program library.

Step 4

Catalog the new USTRE1 to the program library.

Step 5

Test the new exit by invoking USTOR.

If an error exists in the new USTRE1 exit routine, USTOR may terminate abnormally. It is therefore recommended that precautions be taken to retain a usable copy of USTRE1 in order to ensure continued reliable service.

USTRE1 can optionally be link edited as part of the USTOR utility for performance reasons. If not link edited with USTOR, USTRE1 is loaded once during the initialization of USTOR.

Size restrictions are not imposed on USTRE1. Size is a consideration, however, because not only must the exit fit into the same thread in which USTOR executes, but the cataloged region size of USTOR must also consider the size of USTRE1. Note that if there is insufficient storage to load USTRE1, an error will result.

How to Use USTRE1

Upon entry to USTRE1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of Com-plete's COMREG. Word 2 contains the address of the current UPCB. Word 3 contains the address of the current THCB. Word 4 contains the address of the current TIB. Word 5 contains the address of the USTOR command just provided by the terminal user. Word 6 contains the address of a halfword to be used by USTRE1 to pass a return code back to USTOR.

Security is provided by determining if the current user or TIB is permitted to execute the requested USTOR function and setting the return code accordingly.

USTRE1 Conventions

The following table summarizes the USTRE1 linkage conventions.

Feature	Convention
Attributes	None required
Size	Limited only by the USTOR thread size.
Registers at Entry	Register 1 Address of the parameter list Register 13 Address of an 18-word save area Register 14 Return address in USTOR Register 15 Entry point address in USTRE1
Registers at Return	Registers 2 through 13 must be unchanged.
Parameters	Word 1 Address of the COMREG Word 2 Address of the UPCB Word 3 Address of the THCB Word 4 Address of the TIB Word 5 Address of the USTOR function Word 6 Address of a return code halfword
Return Codes	0 Allow the request. 4 Disallow the request. 8 Disallow the request, and terminate USTOR.
Considerations	a Is loaded once per invocation of USTOR. b Can be link edited, or loaded dynamically.

UTMEX1 - Timer User Exit

UTMEX1 is a user-written routine that examines each new timer request that is to be added to the timer SD file by UTIMER.

A request can be rejected by the exit. If this occurs, a 40-byte disallow message must be passed back to the calling routine.

A dummy UTMEX1 module is distributed with the Com-plete system as a member of the distribution and load libraries.

This chapter covers the following topics:

- How to Use UTMEX1
 - UTMEX1 Conventions
-

How to Use UTMEX1

Upon entry to UTMEX1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of a return code halfword. Word 2 of the parameter list contains the address of a timer record in the format of a DSECT defined by the TIMRSD macro. Word 3 of the parameter list contains the address of 40 bytes used by UTMEX1 to pass a return code back to UTIMER.

UTMEX1 Conventions

The following table summarizes the UTMEX1 linkage conventions.

Feature	Convention
Attributes	None required.
Type	Thread.
Size	No restrictions.
Registers at Entry	<p>Register 1 Address of the parameter list</p> <p>Register Address of the caller's 18-fullword save area 13</p> <p>Register Return address in the calling module 14</p> <p>Register Entry point address 15</p>
Registers at Return	Registers 14 through 12 must be unchanged.
Parameters	<p>Word 1 Address of a return code halfword</p> <p>Word 2 Address of the timer record</p> <p>Word 3 Address of the return code</p>
Return Codes	<p>0 Normal return. Allow log service in the timer SD file.</p> <p>4 Do not allow log service in the timer SD file.</p>
Considerations	<p>a Is loaded by UTIMER.</p> <p>b Linkage is dynamic, with COLOAD.</p>

UTMEX2 - Timer Monitor User Exit

UTMEX2 is a user-written routine called by UTIMRM both:

- Every minute;
- For each request that is to be served.

If it is an 'every minute' request, the exit may perform whatever functions it desires, including internal functions, after which flow may continue (return code = 0), or return again to UTMEX2 (return code = 4).

If it is a serve request, the exit may perform any internal functions, modify the request record, and allow or disallow serving.

A dummy UTMEX2 module is distributed with the Com-plete system as a member of the distribution and load libraries.

This chapter covers the following topics:

- How to Use UTMEX2
 - UTMEX2 Conventions
-

How to Use UTMEX2

Upon entry to UTMEX2, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of a return code halfword. Word 2 of the parameter list contains the address of a timer record in the format of a DSECT defined by the TIMRSD macro. Word 3 of the parameter list contains the address of a halfword containing the TIDs of the terminal that is to be logged off by the automatic loggoff function.

Note:

If the contents of the areas pointed by word 2 and word 3 are both binary zero, it is considered to be an 'every minute' exit call. In this case, any service can be filled into the timer record.

UTMEX2 Conventions

The following table summarizes the UTMEX2 linkage conventions.

Feature	Convention
Attributes	Reentrant.
Type	Thread.
Size	No restrictions.
Registers at Entry	<p>Register 1 Address of the parameter list</p> <p>Register 13 Address of the caller's 18-fullword save area</p> <p>Register 14 Return address in the calling module</p> <p>Register 15 Entry point address</p>
Registers at Return	Registers 14 through 12 must be unchanged.
Parameters	<p>Word 1 Address of a return code halfword</p> <p>Word 2 Address of the timer record</p> <p>Word 3 TID of a terminal to be logged off by the automatic logoff function</p>
Return Codes	<p>For serve request/TIB auto logoffs:</p> <p>0 Allow serving request/TIB auto logoff.</p> <p>4 Do not allow serving request/TIB auto logoff.</p> <p>For "every minute" calls:</p> <p>0 Serve request if it exists; then continue.</p> <p>4 Serve request if it exists; then return to UTMEX2.</p>
Considerations	<p>a Is called by UTIMER.</p> <p>b Linkage is dynamic, with COLOAD.</p>

UTMEX3 - Timer Monitor RJE Exit

UTMEX3 is a user-written routine called by UTIMRM at RJE job submission.

This exit has the same parameters and behaves in the same way as UXEEX3 (see the section on UXEEX3 later in this chapter), except that word 2 of the parameter list points to UTMSCOM, the timer common region, instead of UEPDCOM.

A dummy UTMEX3 module is distributed with the Com-plete system as a member of the distribution and load libraries.

This chapter covers the following topics:

- How to Use UTMEX3
 - UTMEX3 Conventions
-

How to Use UTMEX3

Upon entry to UTMEX3, a set of parameters is received in the form of fullword addresses in register 1. Word 1 of the parameter list contains the address of a return code area in which the status of the request is to be indicated. Word 2 of the parameter list contains the address of the UTMSCOM data area.

Word 3 of the parameter list contains the address of the "call-type" byte being passed to UTMEX3; valid character values are:

- S start of job submission before the first statement.
- sp (space) standard call.
- E end of the submission following the last statement.

Note:

If the contents of the areas pointed to by word 2 and word 3 are both binary zero, it is considered to be an 'every minute' exit call. In this case, any service can be filled into the timer record.

Word 4 of the parameter list contains the address of the card image. Word 5 of the parameter list contains the address of the area to be used when inserting statements (return code = 8).

Note that the last two parameters are valid only if the call-type byte is a space.

Upon return to the loading module from UTMEX3, the return code halfword is examined and processed as follows:

- 0 Normal return; the next statement is processed, unless the final statement has been processed.
- 4 Delete the current statement. This return code is valid only if the call-type byte is a space.
- 8 Insert a statement before the current card image statement. The statement to be inserted must be in an area pointed to by the fifth parameter in the parameter list (+16). This return code is valid only for the call-type byte values space and E.
- 12 Abort the RJE submission.

At the end of the job stream, the call type is set to E. UTMEX3 is called repeatedly until a return code of either 0 or 12 is given.

This allows for the insertion of multiple statements at the end of the job stream.

To modify a statement, place it in the statement work area addressed by the fourth parameter in the parameter list (+12), modify the statement there, and return with a return code 0.

UTMEX3 Conventions

The following table summarizes the UTMEX3 linkage conventions.

Feature	Convention								
Attributes	Reentrant.								
Type	Thread.								
Size	No restrictions.								
Registers at Entry	<table> <tr> <td>Register 1</td> <td>Address of the parameter list</td> </tr> <tr> <td>Register 13</td> <td>Address of an 18-fullword save area</td> </tr> <tr> <td>Register 14</td> <td>Return address in the calling module</td> </tr> <tr> <td>Register 15</td> <td>Entry address of UTMEX3</td> </tr> </table>	Register 1	Address of the parameter list	Register 13	Address of an 18-fullword save area	Register 14	Return address in the calling module	Register 15	Entry address of UTMEX3
Register 1	Address of the parameter list								
Register 13	Address of an 18-fullword save area								
Register 14	Return address in the calling module								
Register 15	Entry address of UTMEX3								
Registers at Return	Registers 12 through 14 must be unchanged.								

Feature	Convention	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of the UTMSCOM data area
	Word 3	Address of a flag byte indicating the type of call: S Start of job submission (before first statement) sp (space) standard call E end of the submission (following the last statement)
	Word 4	Address of the card image.
	Word 5	Address of the area to be used when inserting statements (return code = 8)
Return Codes	0	Normal return. Process the next statement unless end-of-file.
	4	Delete the current statement (valid only if call-type =space).
	8	Insert a statement before the current card image. Statement to be inserted must be in an area pointed to by word 5 of the parameter area . This return code is valid only for call-types E and sp (space).
	12	Abort the submission (HASP only).
Considerations	a	Is called by U2TSUB, a subroutine of UTIMRM .
	b	Linkage is dynamic, with COLOAD.

UUEDEX - UED Security Exit

UUEDEX is a user-written routine called by the UED utility program before the execution of certain commands entered by the terminal operator. This module allows you to define security restrictions on the use of the various functions. (See word 2 in the UUEDEX conventions table.)

The UED utility program is a set of logically related modules, each of which services a specific function. Each function requested by a terminal operator is logically processed by a separate module. Each of these modules issues a call to the user-written exit routine UUEDEX before servicing the requested function. Consequently, you can restrict, permit, or eliminate any or all UED functions.

When UED is initially invoked by the terminal operator, the GETCHR function is executed and the information obtained is passed to the user-written exit routine UUEDEX in the form of an address constant passed in the parameter list.

The UUEDEX module is only loaded once per invocation of UED. This implies that internal switches can be set and subsequently referenced. Note, however, that each new invocation of UED loads a new version of UUEDEX, causing the switches to be reset.

A dummy UUEDEX module is distributed with the Com-plete system as a member of the distribution source library and the distribution load library.

Note:

If no additional security is established by the user, only MVS password protection security exists for UED functions.

This chapter covers the following topics:

- How to Use UUEDEX
 - UUEDEX Conventions
-

How to Use UUEDEX

Upon entry to UUEDEX, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of a return code halfword initialized to zero. Word 2 contains the address of the halfword entry code that identifies the primary function being performed. Word 3 contains the address of the GETCHR information area. Word 4 contains the address of the two-character library code table entry, if any, as it exists in the UEDTB1 module (see UEDTB1 Entry DSECT). The UEDTB1 module is described earlier in this chapter.

Additional parameters passed via register 1 are the address of a pseudo-open control block (XOPNCB) for the file being accessed, the address of the UED edit control block (EDITCB), the address of a submit buffer where each statement of a submitted job stream is placed before being submitted, and the address of an insert buffer where a statement is to be placed to enable insertion of statements in a job stream being submitted.

XOPNCB, the pseudo-open control block, provides information such as the name of the file being edited, member name, volume sequence number identifier, etc. The format and contents of this control block are described in UED Pseudo-Open Control Block.

EDITCB, the UED edit control block, provides information such as the command table address, the command buffer address, a pointer to the current command or operand, the address of the SD buffer, etc. The format and contents of this control block are described in UED Edit Control Block.

During processing of the SUBMIT function, UUEDEX is invoked multiple times for the purpose of JCL interrogation and modification. Word 7 of the parameter list points to a submit buffer that contains the current JCL statement for submission. Note that UUEDEX can modify the contents of this buffer.

Word 8 of the parameter list points to an insert buffer used for JCL statement insertion. Upon entry to UUEDEX, byte one of the insert buffer contains a call-type flag to indicate the timing of the call.

During processing of the SUBMIT function, return codes in the first parameter indicate the action to be taken. If a return code 8 is returned, UUEDEX will be called again to allow multiple insertions before any given statement in the submit buffer. Note that a return code 8 is valid only for the call-type values of space and E.

To define security for a specific function, test for the existence of the appropriate function, establish the desired level of authorization, and set the return code in word 1 of the parameter list to indicate acceptance or rejection. This return code is also used to indicate to the SUBMIT function the action to be taken (insert a statement, delete a statement, permit a statement).

UUEDEX Conventions

The following table summarizes the UUEDEX linkage conventions.

Feature	Convention								
Attributes	None required.								
Type	Thread.								
Size	Restricted to the UED thread region								
Registers at Entry	<table border="0"> <tr> <td>Register 1</td> <td>Address of the parameter list</td> </tr> <tr> <td>Register 13</td> <td>Address of an 18-fullword save area</td> </tr> <tr> <td>Register 14</td> <td>Return address in the calling module</td> </tr> <tr> <td>Register 15</td> <td>Entry address within UUEDEX</td> </tr> </table>	Register 1	Address of the parameter list	Register 13	Address of an 18-fullword save area	Register 14	Return address in the calling module	Register 15	Entry address within UUEDEX
Register 1	Address of the parameter list								
Register 13	Address of an 18-fullword save area								
Register 14	Return address in the calling module								
Register 15	Entry address within UUEDEX								
Registers at Return	Registers 2 through 13 must be unchanged.								

Feature	Convention	
Parameters	Word 1	Address of a return code halfword for indicating the status of the request.
		Return codes are indicated in the return code entry of this table.
	Word 2	Address of a halfword entry code: 0 UED initialization requested. 4 READ requested. 8 SAVE requested. 12 SUBMIT requested. 20 UED termination requested.
	Word 3	Address of the storage area containing GETCHR information.
	Word 4	Address of a table entry in UEDTB1 for the two-character code for the library being edited. This address is zero if no code is used.
	Word 5	Address of the pseudo-open DCB (XOPNCB).
	Word 6	Address of the UED Edit Control Block (EDITCB).
	Word 7	Address of the card image being submitted.
Word 8	Address of the card image insert buffer, byte 1 of which contains a call-type flag:	S Start of job submission prior to first statement
		sp Space (statement being submitted)
		E End of submission following last statement

Feature	Convention
Return Codes	<p>SUBMIT processing only:</p> <p>0 Submit from the SUBMIT buffer.</p> <p>4 Delete card from the SUBMIT buffer.</p> <p>8 Submit from the insert buffer.</p> <p>12 Abort the submission. Other functions :</p> <p>Other functions:</p> <p>0 Allow the function.</p> <p>4 Disallow the function.</p>
Considerations	<p>a. References:</p> <p>DSECT XOPNCB OPEN DCB</p> <p>DSECT CMEDTB1 Library Codes</p> <p>DSECT TMGETCHR GETCHR Table</p> <hr/> <p>b. First character of insert buffer:</p> <p>S = Start of job submission prior to first statement</p> <p>sp = Space (statement being submitted)</p> <p>E = End of submission following last statement</p> <hr/> <p>c. Must be link edited with UED</p>

UUMAX1 - UMAP Initialization Exit

UUMAX1 is a user-written routine called by the UMAP utility program before the user receives the UMAP menu. This module allows the user to modify global defaults to UMAP and restrict UMAP execution by user ID.

This chapter covers the following topics:

- How to Use UUMAX1
 - UUMAX1 Conventions
-

How to Use UUMAX1

Upon entry to UUMAX1, a set of parameters is received in the form of fullword addresses in register 1.

UUMAX1 can issue any Com-plete function in the process of determining the security restrictions to be imposed upon a terminal user. Sample functions include:

- COLOAD Load a table containing security information;
- COLINK Invoke another user-written module;
- GETCHR Obtain information about the terminal user;
- WRTC Request additional information;
- READ Obtain the requested additional information.

UUMAX1 Conventions

The following table summarizes the linkage conventions of UUMAX1.

Feature	Convention
Attributes	None required.
Type	Thread.
Size	Restricted to the UMAP thread region
Registers at Entry	<p>Register 1 Address of the parameter list</p> <p>Register 13 Address of an 18-fullword save area</p> <p>Register 14 Return address in the calling module</p> <p>Register 15 Entry address of UUMAX1</p>
Registers at Return	Registers 2 through 13 remain unchanged. Register 15 contains the return code.
Parameters	<p>Word 1 Address of the eight-byte user ID.</p> <p>Word 2 Address of the two-byte TID.</p> <p>Word 3 Address of the 44-byte designated map library file name.</p> <p>Word 4 Address of the six-byte volume serial number.</p> <p>Word 5 Address of the eight-byte user exit communications area. This area is passed to all user exits and is neither modified nor checked by UMAP.</p> <p>Word 6 Address of a two-byte area containing the default constant and variable field indicators.</p>
Return Codes	<p>0 Allow the request.</p> <p>4 Disallow the request.</p>
Considerations	Can be link edited, or loaded dynamically.

UUMAX2 - UMAP Command Exit

UUMAX2 is a user-written routine called by the UMAP utility program before servicing a UMAP option. This module allows the user to modify global defaults to UMAP and restrict UMAP execution by user ID.

This chapter covers the following topics:

- How to Use UUMAX2
- UUMAX2 Conventions

How to Use UUMAX2

Upon entry to UUMAX2, a set of parameters is received in the form of fullword addresses in register 1.

UUMAX2 Conventions

The following table summarizes the linkage conventions of UUMAX2.

Feature	Convention
Attributes	Reentrant if in a resident area
Type	Thread
Size	Restricted to the UMAP thread region
Registers at Entry	Register 1 Address of the parameter list Register 13 Address of an 18-fullword save area Register 14 Return address in the calling module Register 15 Entry address of UUMAX2
Registers at Return	Registers 2 through 13 remain unchanged. Register 15 contains the return code.
Parameters	Word 1 Address of the eight-byte user ID. Word 2 Address of the two-byte TID. Word 3 Address of a one-byte UMAP option code:
Return Codes	0 Allow the request. 4 Disallow the request.
Considerations	Can be link edited, or loaded dynamically.

UUMAX3 - UMAP Termination Exit

UUMAX3 is a user-written routine called by the UMAP utility program before exiting UMAP. This module allows you to force cleanup of SD files, and to control the termination of UMAP.

This chapter covers the following topics:

- How to Use UUMAX3
 - UUMAX3 Conventions
-

How to Use UUMAX3

Upon entry to UUMAX3, a set of parameters is received in the form of fullword addresses pointed to by register 1.

UUMAX3 Conventions

The following table summarizes the linkage conventions of UUMAX3.

Feature	Convention
Attributes	None required.
Type	Thread.
Size	Restricted to the UMAP thread region
Registers at Entry	<p>Register 1 Address of the parameter list</p> <p>Register 13 Address of an 18-fullword save area</p> <p>Register 14 Return address in the calling module</p> <p>Register 15 Entry address of UUMAX3</p>
Registers at Return	Registers 2 through 13 remain unchanged. Register 15 is ignored.
Parameters	<p>Word 1 Address of the eight-byte user ID.</p> <p>Word 2 Address of the two-byte TID.</p> <p>Word 3 List of the five eight-byte SD file name fields. If less than five names are in the list, the list of valid names will be terminated with a HEX '00'.</p> <p>Word 4 Address of the eight-byte user exit communications area. This area is passed to all user exits and is neither modified nor accessed by UMAP.</p>
Return Codes	Ignored.
Considerations	Can be link edited, or loaded dynamically.

UUPDX1 - UPDS Security Exit (MVS Only)

UUPDX1 is a user-written routine called by the UPDS utility program before the execution of any command entered by the terminal operator. This module allows you to define security restrictions on the use of the various functions.

The UPDS utility program is a set of logically related modules, each of which services a specific function. Therefore, each function requested by a terminal operator is logically processed by a separate module. Each of these modules issues a call to the user-written exit routine UUPDX1 before servicing the requested function. Consequently, you can restrict, permit, or eliminate any or all UPDS functions.

When UPDS is initially invoked by the terminal operator, the GETCHR function is executed. The information obtained is passed to the user-written exit routine UUPDX1 in the form of a parameter list address. This information can be further referenced in order to place additional restrictions on the use of UPDS.

Because the UUPDX1 module is only loaded once per invocation of UPDS, internal switches can be set and subsequently referenced. Note, however, that each new invocation of UPDS loads a new version of UUPDX1, causing the switches to be reset.

A dummy UUPDX1 module is distributed with the Com-plete system as a member of the distribution source and load libraries.

Note:

Only MVS password security exists for UPDS functions if you do not establish any. Obviously, SCRATCH and RENAME functions cannot be performed in read-only libraries.

This chapter covers the following topics:

- How to Use UUPDX1
- UUPDX1 Conventions

How to Use UUPDX1

Upon entry to UUPDX1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of a return code halfword initialized to zero. Word 2 contains the address of an area containing the UPDTB1 information table. This table, which passes information to the user-written exit, is described by the UPDTB1 macro (assemble sample UUPDX1 to see the layout of this area) and illustrated in UPDTB1 Information Control Block. Word 3 contains the address of the GETCHR information table. Word 4 contains the address of the library code entry in module UEDTB1, if a two-character library code was used to make the request (see UEDTB1 Entry DSECT). If no library code was entered however, the corresponding field in UEDTB1 contain binary zeros, and the address given in the parameter list must not be used.

To define security for a specific function, test for the existence of the appropriate function, establish the desired level of authorization, and set the return code pointed to by word 1 of the parameter list to indicate acceptance or rejection.

If a UPDS command has been entered that performs a modification to a file (for example, SCRATCH, RENAME), an attempt is made to allocate the file with a disposition of OLD. If this attempt is unsuccessful, an allocation of SHR is attempted. The ENQ of DISP=OLD or DISP=SHR is obtained on the queue element names of SYSDSN and SYSIEWLP; the function requested is only executed if the ENQs are successful.

UUPDX1 Conventions

The following table summarizes the UUPDX1 linkage conventions.

Feature	Convention
Attributes	None required.
Type	Thread.
Size	Restricted by the UPDS thread region size
Registers at Entry	Register 1 Address of the parameter list Register 13 Address of an 18-fullword save area Register 14 Return address in the calling module Register 15 Entry address of UUPDX1
Registers at Return	Registers 2 through 13 must be unchanged.
Parameters	Word 1 Address of a return code halfword for indicating the status of the request. 0 Allow the request. 4 Disallow the request. Word 2 Address of the UPDTB1 table Word 3 Address of the GETCHR information table Word 4 Address of the UEDTB1 entry, if any; otherwise, binary zeros
Return Codes	0 Allow the request. 4 Disallow the request.
Considerations	a Is loaded once per call of UPDS. b Can be link edited with UPDS. c Can be link edited, or dynamically loaded.

UUQEX1 - UQ Security Exit

UUQEX1 is a user-written routine called by the UQ utility program before processing UQ functions requested by the terminal operator. This module defines security restrictions on the use of the various functions.

The UQ utility program is a set of logically related modules, each of which services a specific function (H, R, T, M, O, etc.). Each function requested by a terminal operator is logically processed by a separate module. Each of these modules issues a call to the user-written exit routine UUQEX1 before servicing the requested function. Consequently, you can restrict, permit, or eliminate any or all the UQ functions.

Because the UUQEX1 module is only loaded once per invocation of UQ, internal switches can be set and subsequently referenced. Each new invocation of UQ will load a new version of UUQEX1, causing the switches to be reset.

A dummy UUQEX1 module is distributed with the Com-plete system as a member of the distribution source and load libraries.

Note:

No security exists for UQ functions unless it is established by you..

This chapter covers the following topics:

- How to Use UUQEX1
 - UUQEX1 Conventions
-

How to Use UUQEX1

At entry to UUQEX1, a set of parameters is received in the form of fullword addresses in register 1. Word 1 of the parameter list contains the address of a return code area in which the status of the request is to be indicated. Word 2 of the parameter list contains the address of a code indicating the nature of the function being requested.

To define security for a specific function, test for the existence of the appropriate function code, establish the desired level of authorization, and set the return code to indicate acceptance or rejection.

In some cases, if a function is rejected, no security violation is posted at the terminal; the function is simply suppressed. For example, if a user ID is restricted to viewing only SYSOUT in output class A, a request for the Q function of UQ will display only SYSOUT in class A.

As described in the Com-plete Utilities documentation, the UQ utility program recognizes well-defined comment cards as part of the input job stream in order to further define security. One of the comment cards can be used to pass information to UUQEX1, providing additional security criteria at the user level. This comment statement is in the format:

```
// *UQ USER ...xxx... (MVS)  
* *UQ USER ...xxx... (VSE)
```

where *xxx* can contain any desired information consisting of a maximum of 60 characters. This information is passed to UUQEX1 for all functions issued for the specific job containing the comment statement. With this special JCL comment statement, an installation can impose security not supported by the standard UUQEX1 conventions.

For example, if a password is required for a certain job's SYSOUT/SYSLST to be displayed using the S function of UQ, the user comment statement could be used in order to communicate the password to the exit routine. The exit routine could then prompt the terminal operator for the password and disallow the request if it is not correctly entered.

UUQEX1 Conventions

The following summarizes the UUQEX1 linkage conventions.

Attributes

None required.

Type

Thread.

Size

Restricted to the UQ thread region.

Registers at Entry

Register 1	Address of the parameter list
Register 13	Address of an 18-fullword save area
Register 14	Return address in the calling module
Register 15	Entry address within UUQEX1

Registers at Return

Registers 2 through 13 must be unchanged.

Parameters

Word 1	Address of a return code halfword. The return code is preset according to the security check on the JCL cards before the exit is called. Possible codes:			
	<table> <tr> <td>0</td> <td>Security check passed.</td> </tr> <tr> <td>4</td> <td>Security check failed. This allows the user exit to override the previous security check by resetting return code 4 to 0.</td> </tr> </table>	0	Security check passed.	4
0	Security check passed.			
4	Security check failed. This allows the user exit to override the previous security check by resetting return code 4 to 0.			
Word 2	Address of a UQ function code:			

	<p>A Active display</p> <p>C Cancel request</p> <p>D DASD unit display</p> <p>DE Destination routing request</p> <p>H Hold request</p> <p>K Console command request</p> <p>M Console messages display</p> <p>O Operator reply ID display</p> <p>OC Output class alteration</p> <p>Q Generalized job queue display (once per display request)</p> <p>QL Specific job queue display (once per displayed job)</p> <p>R Release request</p> <p>S Spool display request</p> <p>T Tape display request</p> <p>V DASD volume space display</p>
Word 3	Address of the job name (not applicable for the A, D, K, M, O, Q, T, V, C, W and R functions).
Word 4	Address of the job number; 32-bit unsigned, binary (not applicable for the A, D, K, M, O, Q, T, V, C, W and R functions).
Word 5	Address of the job class code (QL function only). Gives message class in JES2, execution class in JES3.
Word 6	Address of the job queue code (QL function only):
	<p>I Input queue</p> <p>O Output queue</p> <p>X Executing (HASP and JES2 systems only)</p> <p>P Purge active (HASP and JES2 only)</p>
Word 7	Address of the destination code; eight characters, left justified (H, R, C, DE, and QL functions only).
Word 8	Address of the spool ID request type (S function only):

	CC Condition Code JL JCL on input queue SI Sysin file SM System message file SO Sysout file
Word 9	Address of the 60-byte area containing data from the UQ statement (S, H, R, C and DE functions only): <pre>// *UQ USER ... (MVS) * *UQ USER ... (VSE)</pre> The statement must precede all other UQ comment statements in order to be valid

Return Codes

0	Allow the request.
4	Security violation.

Considerations

a	Is loaded once per call execution of UQ.
b	Can be link edited or loaded dynamically.

UUSEX1 - USDLIB Security Exit

UUSEX1 is a user-written routine called by the USDLIB utility program each time before contents of a single SD file are displayed or zapped, or an SD file is deleted. This routine allows you to define security restrictions on the access of SD files. Possible uses include:

- To MESGSW a record recording the access of an SD data set;
- To restrict access to SD files that may contain sensitive information;
- To restrict use of subfunctions, for example, zap or delete.

Because the UUSEX1 module is only loaded once per invocation of USDLIB, internal switches can be set and referred to.

A dummy UUSEX1 module is distributed with the Com-plete system as a member of the distribution source library and the distribution load library.

Note:

No security exists for USDLIB functions unless it is established by you.

This chapter covers the following topics:

- How to Use UUSEX1
 - UUSEX1 Conventions
-

How to Use UUSEX1

Upon entry to UUSEX1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 contains the address of a halfword return code to be initialized in the exit. A value of 0 allows access; a value of 4 disallows access. Word 2 of the parameter list contains the address of a six-byte field containing the name of the SD file being accessed. Word 3 of the parameter list contains the address of a two-byte field containing the TID number of the SD file. A value of binary zeros means that the SD file was created with SHR status. Word 4 of the parameter list contains the address of a one-byte field containing the USDLIB subfunction code.

To define security, check which SD file is being accessed, check the subfunction being executed, and set the return code to indicate acceptance or rejection.

Upon return from UUSEX1, the return code area is examined by USDLIB. If the return code is not zero, the operation is aborted and a security violation message is issued.

UUSEX1 Conventions

The following table summarizes the UUSEX1 linkage conventions.

Feature	Convention	
Attributes	Reentrant if in a resident area.	
Type	Thread.	
Size	Maximum of 2048 bytes	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address of UUSEX1
Registers at Return	All registers must be unchanged.	
Parameters	Word 1	Address of a halfword return code
	Word 2	Address of the eight-byte file name
	Word 3	Address of the two-byte TID for the file; a TID value of 0 indicates that the file has SHR status
	Word 4	Address of the one-byte subfunction code:
	S	Show records of the SD file
	Z	Zap a record of the SD file
	D	Delete the SD file
Return Codes	0	Allow the request.
	4	Security violation.
Considerations	a	Is loaded once per call of USDLIB.
	b	Is loaded before invoking the display request.

UUSPL0 - USPOOL Command Exit

UUSPL0 is a user-written routine called by USPOOL after the terminal user has issued a valid command. UUSPL0 defines security restrictions pertaining to the use of all USPOOL functions and can also be used to restrict access to specific printers.

The UUSPL0 module is loaded once per invocation of USPOOL. This means that internal switches can be set and subsequently referred to. Note, however, that each new invocation of USPOOL causes a new copy of UUSPL0 to be loaded, thereby causing all switches in UUSPL0 to be reset.

A dummy UUSPL0 module is distributed with the Com-plete system as a member of the distribution source and load libraries.

Note:

No security exists for USPOOL functions, unless it is established by you or restricted via Com-plete Security System definitions.

This chapter covers the following topics:

- How to Use UUSPL0
 - UUSPL0 Conventions
-

How to Use UUSPL0

Upon entry to UUSPL0, a set of parameters is received in the form of 5 fullword addresses. For the meaning of the words, see the parameters in the description of the UUSEX1 exit.

Upon return from UUSPL0 to USPOOL, the return code area is checked. If the return code is not zero, the requested operation is aborted and an error message is displayed on the main menu screen.

UUSPL0 Conventions

The following table summarizes the UUSPL0 linkage conventions:

Feature	Convention	
Attributes	Reentrant if in a resident area.	
Type	Thread.	
Size	Restricted to the USPOOL thread region size	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address of UUSPL0
Registers at Return	All registers except R15 must be unchanged.	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of a four byte function code
	Word 3	Address of the requestors specifications
	Word 4	Address of the listqueue entry definitions or address of the printername
	Word 5	Optional for listqueue update functions, address of the old listqueue entry definitions
Return Codes	0	Allow the request.
	4	Disallow the request.

Function Codes passed via word 2:

PO Printer overview

LQ Listqueue overview

MOx Modify Q-Entry x = U update M move C copy P purge

DP Display printout in screen

OPx Operate printer x = S start R reset H halt

Data Area passed via word 3:

USRXP	DSECT	,	
USRXUID	DS	CL8	User name
USRXACC	DS	CL12	Account
USRXAUT	DS	AL2	Authorization code
USRXCTL	DS	CL2	Control/Noncontrol
USRXPL	EQU	*-USRXP	Length of element

Data Area passed via word 4 (and word 5):

OLQLINE	DSECT	,	
OLQNAME	DS	CL8	List name
OLQNUM	DS	AL2	Number
OLQFORM	DS	CL4	Form
OLQSTAT	DS	C	Status
OLQLINS	DS	AL2	Number of lines
OLQCOPY	DS	AL2	Number of copies
OLQPRI	DS	AL2	Priority
OLQUSER	DS	CL8	Originator
OLQLDRV	DS	CL8	Logical output driver
OLQDEST	DS	CL8	Destination
OLQLINEL	EQU	*-OLQLINE	Length of element

UUSVX1 - USERV Security Exit (VSE Only)

UUSVX1 is a user-written routine called by the USERV utility program before the execution of any command entered by the terminal operator. This module allows you to define security restrictions on the use of the various functions.

When USERV is initially invoked by the terminal operator, the GETCHR function is executed and the information obtained is passed to the user-written exit routine UUSVX1 in the form of a parameter list address. This information can be further referenced to place additional restrictions on the use of USERV.

Because the UUSVX1 module is only loaded once per invocation of USERV, internal switches can be set and subsequently referenced. Note that each invocation of USERV loads a new version of UUSVX1, causing the switches to be reset.

A dummy UUSVX1 module is distributed with the Com-plete system as a member of the distribution source and load libraries.

This chapter covers the following topics:

- How to Use UUSVX1
 - UUSVX1 Conventions
-

How to Use UUSVX1

Upon entry to UUSVX1, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of a return code halfword initialized to zero. Word 2 contains the address of an area containing the UPDTB1 information table. This table passes information to the user-written exit. It is described by the UPDTB1 macro (assemble sample UUPDX1 to see the layout of this area) and illustrated in UPDTB1 Information Control Block. Word 3 contains the address of the GETCHR information table. Word 4 contains the address of the library code entry in module UEDTB1 if a two-character library code was used to make the request (see UEDTB1 Entry DSECT). If no library code was entered, the corresponding field in UEDTB1 contains binary zeros and the address given in the parameter list must not be used.

To define security for a specific function, test for the existence of the appropriate function, thereby establishing the desired level of authorization, and set the return code pointed to by Word 1 of the parameter list to indicate acceptance or rejection.

Size restrictions are not imposed on UUSVX1. Size is a consideration, however, since the exit must fit into the same thread in which USERV executes. Because USERV is not threadlocked, it is invoked in one of any available threads. If the thread is too small to contain both USERV and the exit routine UUSVX1 (which is loaded), an error will result.

UUSVX1 Conventions

The following table summarizes the linkage conventions of UUSVX1.

Feature	Convention	
Attributes	Reentrant if in a resident area	
Type	Thread	
Size	Restricted by the USERV thread region	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address of UUSVX1
Registers at Return	Registers 2 through 13 must be unchanged.	
Parameters	Word 1	Address of a return code halfword for indicating the status of the request: 0 Allow the request. 4 Disallow the request.
	Word 2	Address of the UPDTB1 table
	Word 3	Address of the GETCHR information table
	Word 4	Address of the UEDTB1 entry, if any; otherwise binary zeros
Return Codes	0	Allow the request.
	4	Disallow the request.
Considerations	a	Is loaded once per call of USERV.
	b	Is loaded dynamically.

UUTEX1 - UUTIL Security Exit

UUTEX1 is a user-written routine called by UUTIL before a function processor is called. The exit can be used to impose security restrictions on the use of certain UUTIL functions, over and above those already defined by Com-plete.

By default, certain functions of the UUTIL maintenance utility are only open to "Super Users". Control users can access "Super User" functions by either specifying the maintenance password when invoking UUTIL or, alternatively press PF10 on the UUTIL main menu. This prompts for the maintenance password and, when confirmed, will bring up an additional number of utilities usually restricted to the administrator.

This chapter covers the following topics:

- How to Use UUTEX1
 - UUTEX1 Conventions
-

How to Use UUTEX1

The first function of UUTEX1 (function code 0) allows the UUTEX1 user exit to override this password protection and allow a control user to automatically receive the full functionality when invoking UUTIL.

The second function of UUTEX1 (function code 4) allows the exit to control the use of the individual subfunctions of UUTIL.

UUTEX1 Conventions

Feature	Convention		
Registers at Return	Registers 12 through 14 must be unchanged.		
Parameters	Word 1	Address of a halfword function code: 0 Check if the maintenance password is required for control users to enter "super-user" status. 4 Check if UUTIL function should be allowed for the current user.	
Return Codes	For fct. 0	0 - Password required	4 - Password not required
	For fct. 4	0 - Allow the request	4 - Disallow the request
Considerations	a	Called by UUTIL.	
	b	Linkage is dynamic, with COLOAD.	

UXEEX1 - UEDIT Initialization Exit

Note:

A familiarity with the functional design of the UEDIT utility program is necessary for a full understanding of this section (see *UEDIT - Functional Design*).

UXEEX1 defines security restrictions on the use of the various UEDIT functions and restricts user ID access to specific libraries or members. UXEEX1 can be used to:

- Check a user ID for authorization to access the source library being edited;
- MESGSW a record identifying the access of the library or member;
- Use the CAPTUR function to provide an audit trail of the edit operation;
- Exit to an alternative access method routine in order to perform the requested operation;
- Alter the DSN, library code, VOLSER, and/or the file organization.

The UXEEX1 module is only loaded once per invocation of UEDIT. This means that internal switches can be set and subsequently referenced. Remember, however, that each new invocation of UEDIT will load a new version of UXEEX1, causing the switches to be reset.

A dummy UXEEX1 module is distributed with the Com-plete system as a member of the distribution source and load libraries.

Note:

No security exists for UEDIT functions unless it is established by you.

This chapter covers the following topics:

- How to Use UXEEX1
- UXEEX1 Conventions

How to Use UXEEX1

UXEEX1 is a user-written routine called by the UEDIT initialization routine (UEDIT) in one of three ways:

- Upon a direct UEDIT entry call request;
- To set the menu default characteristics;
- After syntax checking the request, but before calling UEBP to perform the requested operation on the source library.

Upon entry to UXEEX1, a set of parameters is received in the form of fullword addresses in register 1. Word 1 of the parameter list contains the address of a return code area in which the status of the request is to be indicated. Word 2 of the parameter list contains the address of the UEPDCOM data area.

Word 3 of the parameter list contains the address of a one-character code that identifies the origin of the call request being issued for UXEEX1. An X'80' in the high-order byte of the address indicates that this is the last parameter being passed (via register 1) to UXEEX1. The character addressed by the third parameter can be one of the following:

- D UXEEX1 is being entered directly with a non-menu invocation of UEDIT and before processing input information "library ID", "option", and "member".
- M UXEEX1 is being entered before generating the UEDIT output menu.
- E UXEEX1 is being entered before fetching UEBP to perform I/O on the source library. In this situation, the BPCODE field within UEPDCOM is valid, and can be tested or altered. During this call, if the library code passed is spaces, the UEDIT system will not use UEDTB1 but will use the DSN and VOLSER fields in UEPDCOM to access the data set.

To define security, proceed as follows:

- Examine the third parameter to determine the origin of the call request;
- Test for the existence of the desired function (via information passed in UEPDCOM);
- Establish the desired level of authorization;
- Set the return code to indicate either acceptance or rejection.

Upon return to the UEDIT module from UXEEX1, the return code area is examined. If the return code is not zero, the edit operation is aborted and the menu page is output with an error message.

UXEEX1 Conventions

The following table summarizes the UXEEX1 linkage conventions.

Feature	Convention	
Attributes	Reentrant if in a resident area.	
Type	Thread.	
Size	Restricted to the UEDIT thread region	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address within UXEEX1
Registers at Return	All registers must be unchanged.	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of the UEPDCOM data area
	Word 3	Address of a one-character call code:
		D Non-menu invocation of UEDIT
		M Menu invocation of UEDIT
		E Prior to UEBP I/O functions
Return Codes	0	Allow the request.
	4	Security violation.
Considerations	a	Is loaded once per call execution of UEDIT.
	b	Is loaded before invoking UEBP. c. May be link edited, or dynamically loaded.

UXEEX2 - UEDIT Command/Termination Exit

Note:

A familiarity with the functional design of the UEDIT utility program is necessary for a full understanding of this section (see *UEDIT - Functional Design*).

UXEEX2 is a user-written routine loaded during full-screen edit initialization by the module UEPDMN, and is given control whenever the terminal user enters a UEDIT command or requests edit termination by pressing CLEAR. When a UEDIT command is entered, the exit is called after UEPDMN has determined that a command has been entered but before the command is identified and processed.

The primary function of this routine is to define security restrictions on the use of the various UEDIT commands and to logically intercept control at UEDIT termination. UXEEX2 can be used to:

- Check a user ID for authorization to issue certain UEDIT commands;
- Extend the UEDIT command language with commands written by the user;
- Pass control to another program or routine upon UEDIT termination.

Because the UXEEX2 module is loaded once per entry to UEPDMN, internal switches cannot be set and subsequently referred to.

A dummy UXEEX2 module is distributed with the Com-plete system as a member of the distribution source library and the distribution load library.

Note:

No security exists for UEDIT functions unless it is established by you.

This chapter covers the following topics:

- How to Use UXEEX2
 - UXEEX2 Conventions
-

How to Use UXEEX2

UXEEX2 receives control before UEPDMN identifies or validates the command. Upon entry to UXEEX2, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 of the parameter list contains the address of a return code area in which the status of the request is to be indicated. Word 2 of the parameter list contains the address of the UEPDCOM data area.

Word 3 of the parameter list contains the address of the "call-type" byte being passed to UXEEX2: if a command has been entered, this field contains a C; otherwise, this field contains an E. When a command is entered, additional fullword parameters (words 4 through 7) give further details about the type of command and its arguments. Word 7 of the parameter list contains an X'80' in the high-order byte of the address, indicating that this is the last parameter in the list.

To define security, test for the existence of the desired command, establish the desired level of authorization, and set the return code to indicate either acceptance or rejection. If the command is acceptable, a return code of 0 will enable execution of the command. Setting the return code to 4 causes the command to be ignored. A return code of 8 invokes the error message "COMMAND NOT PERMITTED".

UXEEX2 Conventions

The following table summarizes the UXEEX2 linkage conventions.

Feature	Convention	
Attributes	Reentrant if in a resident area.	
Type	Thread.	
Size	Restricted to the UEPDMN thread region.	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address of UXEEX2
Registers at Return	All registers must be unchanged.	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of the UEPDCOM data area
	Word 3	Address of a flag byte indicating the type of call: C command E termination
	Word 4	Address of the command verb (valid with command calls only)
	Word 5	Address of a halfword containing the length of the command verb
	Word 6	Address of the command operands
	Word 7	Address of a halfword containing the length of the command operands (0 = no operands)
Return Codes	0	Execute the command.
	4	Ignore the command.
	8	Issue 'command not permitted' diagnostic.
Considerations	a	Is given control for every UEDIT command.
	b	Can be link edited, or dynamically loaded.

UXEEX3 - UEDIT RJE Exit

Note:

A familiarity with the functional design of the UEDIT utility program is necessary for a full understanding of this section (see *UEDIT - Functional Design*).

This exit provides security for the UEDIT utility only. The same function is provided by ULSRRJE on a system-wide basis.

UXEEX3 is a user-written routine loaded during RJE initialization by the module UERJE. It is given control before RJE begins, once for each card image sent to RJE, and after RJE terminates. UXEEX3 can examine, modify, insert, and delete card images in the RJE input stream, as well as abort the JOB submission altogether.

The primary function of this routine is to modify submitted job control statements and, in general, establish control over JCL conventions for submitted jobs. UUEEX3 can be used to:

- Syntax-check JCL;
- Enforce installation standards;
- Insert UQ security statements into the input stream.

Note that any dynamic storage that needs to be acquired by UXEEX3 must be acquired during the first call to the exit, since UEDIT's RJE processor will get and use all available thread storage after the first call.

The UXEEX3 module is loaded once per submit request. This means that internal switches can only be set and subsequently referred to during a single submit request.

A dummy UXEEX3 module is distributed with the Com-plete system as a member of the distribution source and load libraries.

Note:

No security exists for UEDIT functions, unless it is established by you.

This chapter covers the following topics:

- How to Use UXEEX3
 - UXEEX3 Conventions
-

How to Use UXEEX3

Upon entry to UXEEX3, a set of parameters is received in the form of fullword addresses in register 1. Word 1 of the parameter list contains the address of a return code area in which the status of the request is to be indicated. Word 2 of the parameter list contains the address of the UEPDCOM data area.

Word 3 of the parameter list contains the address of the "call-type" byte being passed to UXEEX3; valid character values are:

- S start of job submission before the first statement.
- sp space (statement being submitted).
- E end of the submission following the last statement.

Word 4 of the parameter list contains the address of the statement being submitted. Word 5 of the parameter list contains the address of the area to be used when inserting statements (return code = 8).

Note that the last two parameters are valid only if the call-type byte is a space.

Upon return to UERJE from UXEEX3, the return code halfword is examined and processed as follows:

- 0 Normal return; the next statement is processed, unless the final statement has been processed.
- 4 Delete the current statement. This return code is valid only if the call-type byte is a space.
- 8 Insert a statement before the current card image statement. The statement to be inserted must be in an area pointed to by the fifth parameter in the parameter list (+16). This return code is valid only for the call-type byte values space and E.
- 12 Abort the RJE submission.

At the end of the job stream, the call type is set to E. UXEEX3 is called repeatedly until a return code of either 0 or 12 is given.

This allows for the insertion of multiple statements at the end of the job stream.

To modify a statement, place it in the statement work area addressed by the fourth parameter in the parameter list (+12), modify the statement there, and return with a return code 0.

Once a return code of 12 is given by UXEEX3, UXEEX3 is not called again. Return is to the UEDIT menu page with the error message "SUBMIT ABORTED".

UXEEX3 Conventions

The following table summarizes the UXEEX3 linkage conventions.

Feature	Convention								
Attributes	Reentrant if in a resident area.								
Type	Thread.								
Size	Restricted to the UERJE thread region								
Registers at Entry	<table style="width: 100%; border: none;"> <tr> <td style="width: 20%;">Register 1</td> <td>Address of the parameter list</td> </tr> <tr> <td>Register 13</td> <td>Address of an 18-fullword save area</td> </tr> <tr> <td>Register 14</td> <td>Return address in the calling module</td> </tr> <tr> <td>Register 15</td> <td>Entry address of UXEEX3</td> </tr> </table>	Register 1	Address of the parameter list	Register 13	Address of an 18-fullword save area	Register 14	Return address in the calling module	Register 15	Entry address of UXEEX3
Register 1	Address of the parameter list								
Register 13	Address of an 18-fullword save area								
Register 14	Return address in the calling module								
Register 15	Entry address of UXEEX3								

Feature	Convention	
Registers at Return	All registers must be unchanged.	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of the UEPDCOM data area
	Word 3	Address of a flag byte indicating the type of call: S start of job submission (before first statement) sp (space) statement being submitted E end of the submission (following the last statement)
	Word 4	Address of the statement being submitted.
	Word 5	Address of the area to be used when inserting statements (return code = 8)
Return Codes	0	Normal return. Process the next statement unless end-of-file.
	4	Delete the current statement (valid only if call-type = space).
	8	Insert a statement before the current card image. State-ment to be inserted must be in an area pointed to by word 5 of the parameter area . This return code is valid only for call-types E and sp (space).
	12	Abort the submission.
Considerations	a	Is loaded once per RJE initialization by UERJE.
	b	Is called before the RJE begins for each card image sent to RJE, and after the RJE is finished.
	c	Can be link edited, or loaded dynamically.

UXEEX4 - UEDIT LIBRARIAN/PANVALET Exit

Note:

A familiarity with the functional design of the UEDIT utility program is necessary for a full understanding of this section (see the chapter entitled UEDIT - Functional Design).

UXEEX4 is a user-written routine that allows you to enforce installation standards for the LIBRARIAN "-SEL" and "-ADD" statements and the PANVALET "++ADD" and "++UPDATE" statements. When the UEDIT SAVE operation is executed, control is passed to UXEEX4 for each of these statements existing in a member. This routine can be included:

- In the link edit of the UEDIT subprogram UEBP;
- As an operating system or Com-plete-resident program;
- On the Com-plete program library.

If it is present as a resident program, UXEEX4 must be reentrant. An example of this exit is provided on the distributed source library COM.SOURCE.

This chapter covers the following topics:

- How to Use UXEEX4
 - UXEEX4 Conventions
-

How to Use UXEEX4

Upon entry to UXEEX4, a set of parameters is received in the form of fullword addresses pointed to by register 1. Word 1 contains the address of a halfword return code area where the status of the request is placed. Word 2 contains the address of the UEPDCOM data area. Word 3 contains the address of an 80-byte "++ADD", "++UPDATE", "-ADD", or "-SEL" statement that can be examined and modified. Word 4 contains the address of the Edit Control Block (CMEDCB) data area. Word 5 contains the address of a storage area containing GETCHR information.

Upon return from UXEEX4, the return code halfword is examined and processed as follows:

- 0 The SAVE operation continues.
- 4 The SAVE operation is aborted, and an error message is output on the UEDIT menu display.

UXEEX4 Conventions

The following table summarizes the UXEEX4 linkage conventions.

Feature	Convention	
Attributes	Reentrant if in a resident area.	
Type	Thread.	
Size	Restricted to the UEBP thread region.	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address within UXEEX4
Registers	All registers must remain unchanged. at Return	
Parameters	Word 1	Address of a return code halfword
	Word 2	Address of the UEPDCOM data area
	Word 3	Address of the 80-byte "++ADD", "++UPDATE", "-SEL", or "-ADD" control statement
	Word 4	Address of the EDCB data area
	Word 5	Address of the UEDTMGET data area (GETCHR information)
Return Codes	0	Execute the SAVE command.
	4	Abort the SAVE command.
Considerations	a	Is given control for each SAVE command issued when saving to a LIBRARIAN or PANVALET library.
	b	Can be link edited, or loaded dynamically.

UXEEX5 - Locate Exit

UXEEX5 is a user-written routine that enables users to allow an automatic recall of a library that has been migrated by Hierarchical Storage Manager. When UEDIT or UPDS issue a LOCATE macro, control is passed to UXEEX5, which can then interrogate the volser returned from the catalog and issue the recall SVC if required. An example of this exit is provided on the distribution source library.

This chapter covers the following topics:

- How to Use UXEEX5
- UXEEX5 Conventions

How to Use UXEEX5

Upon entry, a set of parameters is received in the form of fullword addresses pointed to by register 1. Words 1 and 3 are unused. Word 2 contains the address of the data set name (DSN) and word 4 contains the address of the volser returned by the LOCATE macro.

UXEEX5 Conventions

The following table summarizes the UXEEX5 conventions.

Feature	Convention	
Attributes	Reentrant if in a resident area.	
Type	Thread.	
Size	Restricted to the region area.	
Registers at Entry	Register 1	Address of the parameter list
	Register 13	Address of an 18-fullword save area
	Register 14	Return address in the calling module
	Register 15	Entry address within UXEEX5
Registers at Return	All registers must remain unchanged.	
Parameters	Word 1	Unused
	Word 2	Address of the DSN
	Word 3	Unused
	Word 4	Address of a data area. Volser is at +6 in this data area.

Batch Utility Programs

Com-plete's batch utilities enable the system programmer to monitor, operate, and maintain the Com-plete system. This part of the Com-plete System Programming documentation describes the use of the batch utilities, and provides the following information:

- A description of how to use the utility and the job control required for execution
- The control card input, if applicable
- The PARM/SYSPARM input, if applicable
- A DD/DLBL name table describing the DD/DLBL statements used
- A conventions table summarizing the required coding conventions for the utility

The following table summarizes Com-plete's batch utilities.

- **TUBATEST** Provides the means to test the Com-plete Batch interface functions.
- **TUDUMP** Provides the means to obtain hard copy printouts of dump entries in the online dump file of the COMSD data set.
- **TUFILE** Enables the switching of a user file from ONLN to BTCH or vice versa (MVS only).
- **TULIB** Enables the ULIB CATALOG and DELETE commands to be performed in a batch environment.
- **TUMSUTIL** Enables the printing, backup, restoration and information display of selected printout spool files.
- **TUSACAPT** Capture file initialization routine.
- **TUSDUTIL** Enables initialization of the SD data set, backup and restoration of SD files.

TUBATEST - Batch Interface Test Program

The batch utility program TUBATEST allows the functions available via the Com-plete Batch interface to be tested individually.

- How to use TUBATEST
- Control Card Input

How to use TUBATEST

Sample job control can be found in the Com-plete distribution source library in member JCLTUBAT, and looks as follows:

MVS:

```
//JOBTUBA JOB.... JOB CARD INFORMATION ..
//*
//* THIS IS A SAMPLE JOB TO RUN THE TUBATEST UTILITY
//* TO PERFORM TEST FUNCTIONS FOR COMPLETE BATCH FUNCTIONS.
//*
//* THE FOLLOWING CHANGES HAVE TO BE PERFORMED BEFORE RUNNING THIS JOB:-
//*
//* 1. INSERT A VALID JOBCARD.
//* 2. CHECK THE STEPLIB DATASET NAMES.
//*
//* NOTE:-
//* THE FOLLOWING ONLY REQUIRED IF YOU REQUIRE TO OVERRIDE ACSTAB
//* OR IF AN ACSTAB MODULE DOES NOT EXIST IN THE USER.LOAD DATASET.
//*
//* 3. CHANGE nnnn AND sss TO THE ACCESS NODE-ID AND ADABAS ROUTER SVC
//* IDENTIFYING THE TARGET SYSTEM.
//*
//BATCH PROC
//BATCH EXEC PGM=TUBATEST
//STEPLIB DD DSN=COM.LOAD,DISP=SHR
//          DD DSN=COM.USER.LOAD,DISP=SHR      <- FOR ACSTAB
//          DD DSN=ADABAS.LOAD,DISP=SHR      <- FOR ADALNK
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//*COMBTCH DD DSN=NODEnnnn.SVCsss,DISP=(NEW,DELETE),UNIT=SYSDA,
//*          SPACE=(TRK,(1,0,0))
//          PEND
//T1 EXEC BATCH
//BATCH.SYSIN DD *
< control cards >
/*
```

VSE:

```
* $$ JOB JNM=JOBTUBA,DISP=D,CLASS=0
* $$ LST DISP=D,CLASS=A
//JOBTUBATEST JOB.... JOB CARD INFORMATION ..
/*
/* THIS IS A SAMPLE JOB TO RUN THE TUBATEST UTILITY
/* TO PERFORM TEST FUNCTIONS FOR COMPLETE BATCH FUNCTIONS.
/*
```

```

/* THE FOLLOWING CHANGES HAVE TO BE PERFORMED BEFORE RUNNING THIS JOB:-
/*
/* 1. INSERT A VALID JOBCARD.
/* 2. CHECK THE LIBDEF STATEMENTS.
/*
/* NOTE:-
/* THE FOLLOWING ONLY REQUIRED IF YOU REQUIRE TO OVERRIDE ACSTAB
/* OR IF AN ACSTAB MODULE DOES NOT EXIST IN THE USER.LOAD DATASET.
/*
/* 3. CHANGE nnnn AND sss TO THE ACCESS NODE-ID AND ADABAS ROUTER SVC
/* IDENTIFYING THE TARGET SYSTEM.
/*
// LIBDEF *,SEARCH=(SAGLIB.COMUSER,SAGLIB.COMvrs,SAGLIB.ADAvrs)
/* DLBL COMBTCH,'NODEnnnn.SVCsss'
// EXEC PGM=TUBATEST
   GETCHR
/*
/&
* $$ EOJ

```

Control Card Input

The control statements used as input from the SYSIN file determine which functions TUBATEST performs.

The control statement cards are free format . However, the whole of an individual statement must fit onto one 80 character input string.

The general format of the control cards is as follows:

```
<function> <operand1> <operand2> .... <operandn>
```

The number of operands which can be supplied is dependent on the function to be performed. A description of the functions and operands supported is given below:

ABEND

The batch job is terminated with a S007 abend, the online transaction in the target Com-plete is terminated normally.

CAPTUR

GETCHR

EOJ

The online transaction in the target Com-plete is terminated normally, the batch job continues, and if other input cards are available, the requested functions are performed.

Note:

If the TUBATEST is terminated without issuing an EOJ, the on-line transaction will remain in session in the target Com-plete until other action is taken (e.g. Autologoff or operator action).

MESGSW <destination> <message text>

Default destination is VIRTP1. Only one destination can be specified. Text length is 80 bytes.
Default text is blanks. Message class is always 1.

PSOPEN <destination>

Default destination is VIRTP1. Only one destination can be specified. Listname is always ZUBATEST Form is always blanks. Length is always 81.

PSPUT <printout text>

Text length is 80 bytes. Carriage control is ' '.

PSCLOS**RJE <option> <jcl card text>**

Use option H for hold, any other value will release the JCL to the internal reader.

SDOPEN <SD file name>

Default SD filename is ZUBATEST Tibname is always SHR. Number of records is 100. Record length is always 6 Kbytes.

SDCLOS

SD filename is ZUBATEST. Tibname is always SHR.

SDDEL

SD filename is ZUBATEST. Tibname is always SHR.

SDWRT <record text>

SD filename is ZUBATEST. Tibname is always SHR. The SD file is written sequentially.

SDREAD

SD filename is ZUBATEST. Tibname is always SHR. The SD file is read sequentially from the beginning.

After calling the Batch function TUBATEST will, where appropriate, print a display of any relevant control blocks and/or data areas and also display the response code returned from the batch interface routine.

TUDUMP - Dump Print Utility Program

The batch utility program TUDUMP provides the means to obtain hard copy printouts of dump entries in the online dump file of the COMSD data set. Sample execution job control is given in the Com-plete distribution source library in member JCLTUDUM.

When an online program abnormally terminates, it generates a dump entry in the online dump file of the COMSD data set. These dump entries may be reviewed interactively by use of the utility program UDUMP. To obtain a printout of one or more specific dump entries, execute the batch program TUDUMP, or use the PRINT option of UDUMP.

- How to Use TUDUMP
- Parameter Input
- TUDUMP Conventions

How to Use TUDUMP

The job control required to execute program TUDUMP is illustrated below:

```
//TUDUMP      JOB      ... job-card information ...
//TUDUMP      EXEC     PGM=TUDUMP,REGION=512K,
//  PARM='D=mmdd1-mmdd2,I=tid,N=nn1-nn2,P=xxxxxx,L=nn,
//          T=hmmss1-hmmss2,SHORT'
//STEPLIB     DD       DSN=COM.LOAD,DISP=SHR
//SYSPRINT    DD       SYSOUT=X
//SYSUDUMP    DD       SYSOUT=X
//COMSD       DD       DSN=COM.SD,DISP=SHR

* $$ JOB JNM=TUDUMP,DISP=D,CLASS=?      ..... POWER JOB CARD INFORMATION
* $$ LST DISP=D,CLASS=A
// JOB      TUDUMP      ..... JOB CARD INFORMATION
// LIBDEF   PHASE,SEARCH=SAGLIB.COMvrs      <---- Note 1
// DLBL     COMCAT,'????????',,VSAM        <---- Note 2
// DLBL     COMSD,'COM.VSAM.SDFILE',,VSAM,CAT=COMCAT
// EXEC     TUDUMP,SIZE=AUTO,PARM='D=mmdd1-mmdd2,I=tid,N=nn1-nn2,      *
//          P=xxxxxx,L=nn,T=hmmss1-hmmss2,SHORT,
/*
/&
* $$ EOJ
```

1. vrs in these cases relates to the Version, Release and SM level of the Com-plete being installed.
2. Change ???????? to the required catalog file name.

Descriptions of the DD/DLBL statements used in the preceding job control are given in the table in the next section.

Parameter Input

The specific dump entry in the online dump file for which a printout is desired is identified to TUDUMP via parameter information. The format of the parameter information is:

```
D=mmdd1-mmdd2,I=tid,L=nn,N=nn1-nn2,P=xxxxxx,T=hmmss1-hmmss2,
SHORT
```

All parameters are optional, as is their sequence. No parameter input means print the entire dump file.

The arguments are:

D=mmdd1-mmdd2	Default: D=0101-1231
	Specifies a range (mmdd1 through mmdd2) of the date a dump entry was generated. The date is the date as it appears in the UDUMP ALL display. Note that mmdd1 and mmdd2 are three- to four-digit numbers in the form month (one- to two-digits), day (two digits).
I=tid	Default: I=ALL
	Specifies the tid of the terminal causing the dump entry to be generated. Note that tid is a one- to three-digit number.
LINECNT=nn	Default: L=55
	Specifies the number of lines to be printed on each page of output. Note that nn must be between 1 and 99.
N=nn1-nn2	Default: N=001-032
	Specifies the number range (nn1 through nn2) of the dumps as they exist in the online dump file. Either a range or a specific dump number may be specified.
P=xxxxxx	Default: P=ALL
	Specifies the name of the online program that abnormally terminated, causing the dump entry to be generated. The name xxxxxx is the name of the program as it appears in the UDUMP ALL display. To print all programs, specify ALL. To print only programs starting with a "U", specify UTI. To print all programs without "U" in the first position of their name, specify NOUTI.
T=hmmss1-	Default: T=000000-235959hmmss2
	Specifies a range (hmmss1 through hmmss2) of time when a dump entry has been generated. The time is the time as it appears in the UDUMP ALL display. Note that hmmss1 and hmmss2 are five- to six-digit numbers in the form hour (one- to two-digits), minutes (two digits), seconds (two digits).
SHORT	Indicates that only the regs page(s) is to be printed.

The following table summarizes the TUDUMP DD/DLBL names.

DD/DLBL	Description
SYSPRINT/SYSLST	Output file for error messages and for generating the printout of the dump MVS Default DCB: RECFM=FA, LRECL=133, BLKSIZE=133 VSE DEVADDR=SYSLST.
COMSD	SD library MVS Default DCB: None.

TUDUMP Conventions

The following table summarizes the TUDUMP coding conventions.

Feature	Convention														
SYSIN/SYSIPT	None														
PARM/SYSPARM	Format: "D=mmdd1-mmdd2,I=tid,L=nn,N=nn1-nn2, P=xxxxxx,T=hmmss1-hmmss2,SHORT"														
	<table> <tbody> <tr> <td>D=mmdd1-mmdd2</td> <td>Range of date</td> </tr> <tr> <td>I=tid</td> <td>Terminal ID</td> </tr> <tr> <td>L=nn</td> <td>Number of lines per page (1 to 99)</td> </tr> <tr> <td>N=nn1-nn2</td> <td>Range of online dump numbers</td> </tr> <tr> <td>P=xxxxxx</td> <td>Program name</td> </tr> <tr> <td>T=hmmss1-hmmss2</td> <td>Range of time</td> </tr> <tr> <td>SHORT</td> <td>Indicator: print regs page(s) only</td> </tr> </tbody> </table>	D=mmdd1-mmdd2	Range of date	I=tid	Terminal ID	L=nn	Number of lines per page (1 to 99)	N=nn1-nn2	Range of online dump numbers	P=xxxxxx	Program name	T=hmmss1-hmmss2	Range of time	SHORT	Indicator: print regs page(s) only
D=mmdd1-mmdd2	Range of date														
I=tid	Terminal ID														
L=nn	Number of lines per page (1 to 99)														
N=nn1-nn2	Range of online dump numbers														
P=xxxxxx	Program name														
T=hmmss1-hmmss2	Range of time														
SHORT	Indicator: print regs page(s) only														
User Exits	TUDUEX1														
Return Codes	<table> <tbody> <tr> <td>00</td> <td>Normal completion</td> </tr> <tr> <td>04</td> <td>Dump not found</td> </tr> </tbody> </table>	00	Normal completion	04	Dump not found										
00	Normal completion														
04	Dump not found														

TUFILe - File Status (ONLN/BTCH) Switching Facility (MVS only)

TUFILe enables you to switch the status of a user file in a currently running Com-plete from ONLN to BTCH or vice versa. Please refer to the description of online utility UUTIL FM for details about switching a file's status.

- How to Use TUFILe
- Parameter Input
- Condition Codes / Return Codes

How to Use TUFILe

TUFILe can be invoked in two different ways:

- Directly as a BATCH job step using JCL illustrated in sample source library member JCLTUFIL.
- Load and call TUFILe from any other BATCH program. In this case, the calling program must provide parameters in the same format as the operating system does provide them to a program called from a JCL EXEC statement: Register 1 contains the address of a fullword which, in turn, contains the address of a length halfword. This halfword contains the length of the parameter string which is located immediately following this halfword:

R1	contains address of APARM
APARM	DC - A(PARM)
PARM	DC - H'length'DC - CL(length)'parameters'

TUFILe is serially reusable, so even if you call it multiple times, it needs to be loaded only once.

Parameter Input

TUFILe expects 3 parameters, which must be separated by commas:

- The name of the target Com-plete as defined in your ACSTAB (DEST= parameter of an ACSTABLE statement). If you omit this parameter (if the parameter string starts with a comma), the standard rules for defining the target Com-plete take effect as described in the description of *Batch*.
- The desired file status, ONLN or BTCH;
- The DD name of the file.

Example:

PARM= ' COMPLETE , BTCH , MYFILE1 '

Condition Codes / Return Codes

0	Function executed successfully.
4	Connected to Com-plete, but function failed.
8	An error occurred while connecting to Com-plete.
16	Parameter(s) invalid or missing.

If your job contains a SYSPRINT DD statement, all messages received from Com-plete will be logged to SYSPRINT.

TULIB - Program Catalog Maintenance Utility

The purpose of batch utility program TULIB is to enable mass sequential invocation of ULIB commands. It enables three of the functions of the online utility program ULIB to be performed in a batch environment:

- CAT
- DEL
- REF

Note that the remaining functions of ULIB are not supported by TULIB. Sample execution job control is given in the distribution Com-plete source library in member JCLTULIB.

Note:

To use this utility, Com-plete must be active with sysparm SUBSYS-ACTIVATE=ACCESS and defined in ACSTAB.

- How to Use TULIB
- Control Card Input
- TULIB Conventions

How to Use TULIB

The job control required to execute program TULIB is illustrated below:

```
//TULIB      JOB    job-card information
//TULIB      EXEC  PGM=TULIB
//STEPLIB   DD    DSN=COM.USER.LOAD,DISP=SHR
//          DD    DSN=COM.LOAD,DISP=SHR
//          DD    DSN=ADABAS.LOAD,DISP=SHR
//SYSPRINT  DD    SYSOUT=A
//SYSIN     DD    *
DEL,PGMA
CAT,PGMA, RG=24K, TG=THREADSX, PG=TASKY
/*

* $$ JOB JNM=TULIB,DISP=D,CLASS=? ..... POWER JOB CARD INFORMATION
* $$ LST DISP=D,CLASS=A
// JOB    TULIB                ..... JOB CARD INFORMATION
/*
/* =====
/*
/* Com-plete MUST BE ACTIVE WITH ACCESS SUPPORT
/*
/* =====
/*
// LIBDEF   PHASE,SEARCH=(SAGLIB.COMUSER,          /* load ACSTAB */          *
              SAGLIB.COMvrS,                       <---- Note 1          *
```

```

                SAGLIB.CADAvrs),temp
// EXEC      TULIB,SIZE=AUTO
CAT,DEMO,RG=8K
DEL,RMPGMA
/*
/&
* $$ EOJ

```

<---- Note 1

vrs in these cases relates to the Version, Release and SM level of the Com-plete being installed.

Descriptions of the DD/DLBL statements used in the preceding job control are given on the next page.

Control Card Input

The control statements used as input by the SYSIN/SYSIPT file define which function to perform and the programs on which to perform that function.

The format of the control cards used as input to TULIB is exactly the same as for the ULIB, CAT, and DEL commands:

Column 1	CAT, DEL or REF
Column 5	Name of program followed by the desired information as described in the Com-plete Utilities documentation

Note that control statements may not be continued, and that there must be only one function specified per control statement. An unlimited number of control statements may be used.

The following table summarizes the TULIB DD/DLBL names.

DD/DLBL	Description
SYSPRINT/SYSLST	Listing file MVS Default DCB: RECFM=FBA, LRECL=133 VSE DEVADDR=SYSLST
SYSIN/SYSIPT	Sequential input file for control statements MVS Default DCB: None. VSE DEVADDR=SYSIPT

TULIB Conventions

The following table summarizes the TULIB coding conventions.

Feature	Convention
SYSIN/SYSIPT	Sequential input statements: Column 1: CAT, DEL or REF Column 5: Program name, parameters
PARM SYSPARM	None
User Exits	None
Return Codes	00 Normal completion. 04 One or more catalog requests failed 08 Node/SVC wrong 16 Error in Access-Interface

TUMSUTIL - Printout Spool Files Print Utility, COMSPL, Backup And Restore Utility

The batch utility TUMSUTIL provides the following facilities:

- Print of selected printout spool files on the system printer;
- Backup of selected printout spool files onto any sequential data set BACKUP;
- Restore of selected printout spool files from the BACKUP data set into COMSPL;
- List and additional information of selected printout spool files in the COMSPL on the system printer;
- List and additional information of selected printout spool files in the BACKUP data set on the system printer.
- Initialization of COMSPL.

TUMSUTIL can be run in parallel with the Com-plete nucleus for print, backup and list requests (for the restore request there is a restriction). Any printout spool file which is stored in COMSPL according to the data organization of the COMSPL may be processed.

Any printout spool file may be selected using the SYSIN control file. However, in the case of print, backup and restore requests, the request will be terminated for the erroneous part of the printout spool file.

- Additional Details of TUMSUTIL Facilities
- Parameter Input
- How To Use TUMSUTIL
- Control Card Input
- Description of the Options in Detail
- Examples
- TUMSUTIL conventions

Additional Details of TUMSUTIL Facilities

The list directories on the COMSPL and BACKUP data sets provide the following information:

- List name
- Form

- Disposition
- Account of lines
- Account of copies
- Sender (user ID)
- Output formatting routine (logical driver)
- Printer's logical name or its terminal ID
- Date printout was created
- Time printout was created

Print of printout spool files from COMSPL: each request is separated by a header page for easy identification of the beginning and end of each printout spool file being printed.

Parameter Input

If the header is not to be printed, PARM="NHDR" must be specified in the TUMSUTIL EXEC statement.

With PARM="SCAN", it is possible to check the syntax of the SYSIN control statements against the syntax description of this document for all requests.

The following table summarizes the TUMSUTIL DD/DLBL/TLBL names:

Note:

It is not possible to restore a backup of a spool file with a record size of "n" to a spool dataset with a record size of less than "n".

DD/DLBL/TLBL	Description
SYSLST	Listing file for the printing of the TUMSUTIL control statements and messages MVS default DCB: None VSE DEVADDR=SYSLST
SYSPRINT/SYS003	Printer file for the printing of the printout spool files MVS default DCB: RECFM=FBA,LRECL=133 VSE DEVADDR=SYS003
COMSPL	COMSPL file MVS default DCB: none
SYSIN/SYSIPT	Sequential file used for the TUMSUTIL control statements MVS default DCB: LRECL=80 VSE DEVADDR=SYSIPT
BACKUP	Sequential file onto which printout spool files are backed-up. MVS default DCB: LRECL=n*,RECFM=FB VSE DEVADDR=SYS002,RECSIZE=n*
RESTORE	Sequential file from which printout spool files are restored. MVS default DCB: LRECL=n* VSE DEVADDR=SYS001,RECSIZE=n*

* "n" represents the record size of the Com-plete spool data set +4.

How To Use TUMSUTIL

The job control language required to execute TUMSUTIL is illustrated below:

```
//TUMSUTIL      JOB ...job card information...
//TUMSUTIL      EXEC PGM=TUMSUTIL,
//  PARM='SCAN,NHDR'
//STEPLIB      DD DSN=COM.LOAD,DISP=SHR
//SYSLST       DD SYSOUT=A,DCB(BLKSIZE=133)
//COMSPL       DD DSN=COM.SPOOL,DISP=SHR
//BACKUP       DD DSN=...,DCB=(LRECL=n)
//RESTORE      DD DSN=...,DCB=(LRECL=n)
//SYSIN        DD *
               control card input
/*
               (TUMSUTIL Sample VSE Job Control)
* $$ JOB JNM=TUMSUTIL,CLASS=...,DISP=D,LDEST=(,...),PDEST=(,...)
* $$ LST CLASS=...,DISP=D
// JOB TUMSUTIL PO UTILITY
// OPTION      SYSPARM='PARM'
*
* ***      TLBL or DLBL for RESTORE
*
// DLBL       RESTORE,'...',0,SD
// EXTENT     SYS001, ...,1,0, ...,...
// ASSGN      SYS001,DISK,VOL=...,SHR          ASSIGN FOR RESTORE
*
//           TLBL   RESTORE,'...'
//           ASSGN  SYS001,TAPE, ...          ASSIGN FOR RESTORE
*
```

```

* ***      TLBL or DLBL for BACKUP
*
// DLBL    BACKUP,'...',0,SD
// EXTENT  SYS002,...,1,0,...,...
// ASSGN   SYS002,DISK,VOL=...,SHR          ASSIGN FOR BACKUP
*
//        TLBL    BACKUP,'...'
//        ASSGN   SYS002,TAPE,...          ASSIGN FOR BACKUP
*
* ***      ASSGN statement for SYSPRINT
*              (second printer of the partition or SYSLST)
// ASSGN   SYS003,...          ASSIGN FOR SYSPRINT
* $$ LST DISP=D,CLASS=A,LST=SYS003
*
* ***      DLBL for Com-plete SPOOL DATASET
*              (see Com-plete installation specifications)
// DLBL    COMSPL,'...',,VSAM
*
*
// EXEC TUMSUTIL,PARM='...'
      control card input
/*
/*
/&
* $$ EOJ

```

A description of the DD/DLBL/TLBL statements is given in the preceding section **Parameter Input** for TUMSUTIL .

Control Card Input

The control card statements used as input from the SYSIN file select the printout spool files to be processed.

Control statement input is required.

The format of each control statement is:

Command, option(s)

Six commands can be used on the control cards to generate the desired request:

INIT	to initialize the Com-plete spool data set.
PRINT	to print the printout spool files in the COMSPL that meet the selection criteria specified by the options.
LDIR	to print a list of the printout spool files in the COMSPL that meet the selection criteria specified by the options.
B	to backup the printout spool files in the COMSPL that meet the selection criteria specified by the options.
LBKP	to print a list of the printout spool files in the BACKUP data set that meet the selection criteria specified by the options.
R	to restore the printout spool files in the BACKUP data set that meet the selection criteria specified by the options.

Note:

When Com-plete is active without batch support, restore is impossible.

The INIT command is used to initialize a Com-plete spool data set. No options may be supplied with the INIT command, instead the number of records to be written is specified via the RECS keyword on the INIT statement itself for example:

```
INIT RECS=1500
```

The RECS keyword should specify the total number of records which can be written to the dataset, this is given by (HI-ALLOC-RBA / CISIZE), these values can be obtained from an IDCAMS list of message dataset after allocation.

During initialization, TUMSUTIL will set all records in the message dataset to zero and write a control record containing information about the dataset. This information contains, for instance, the maximum number of printouts which can be contained in the dataset. This is currently set to be the number of records in the dataset divided by 8.

If it is required to change this value then the MAXPO parameter in the Com-plete sysparms must be changed to the required value and a Com-plete PO and MSG COLD-START performed.

The options for other commands may be specified in any combination except that a list number never appears in a list backup request (see description of LNUM option); also there is a restriction on the date and time option (see description of DAT1/2 and TIM1/2 options).

At least one option must be specified.

The format of the options specified in the TUMSUTIL control statements is illustrated below (default values are underlined>):

COMP	MULTI, <u>SINGLE</u>
LNAM	cccccccc (8 characters)
LNUM	nnnnnnnn (8 digits)
FORM	cccc (4 characters)
DISP	D,L,H, <u><blank></u>
SUID	cccccccc (8 characters)
LDRV	cccccccc (8 characters)
DEST	cccccccc (8 characters)
DAT1	dd.mm.yy (European), "from" date
	mm/dd/yy (American), "to" date
DAT2	dd.mm.yy (European), "from" date
	mm/dd/yy (American), "to" date
Tim1	hh:mm, "from" time
Tim2	hh:mm, "to" time

Description of the Options in Detail

COMP=MULTI, SINGLE

This option is not used for LBKP requests.

"MULTI" indicates that Com-plete is active and "SINGLE" that Com-plete is not active.

Note that if Com-plete is active and has batch support, COMP="MULTI" is mandatory.

LNAM=cccccccc

Selects a specific printout spool file with this list name.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "cccccccc" is the 1 - 8 byte alphanumeric name that the user assigned to the printout spool file when it was created.

LNUM=nnnnnnnn

Selects a specific printout spool file with this list number.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified. LNUM is never specified on a list backup request.

Note that "nnnnnnnn" is the 1 - 8 digit number that Com-plete assigned to the printout spool file when it was created.

FORM=cccc

Selects a specific printout spool file with this form name.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "cccc" is a 1 - 4 byte alphanumeric form name.

DISP=D,L,H, <blank>

Selects a specific printout spool file with this disposition.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "D,L,H,<blank>" is a 1 byte disposition status, where:

D	delete and print
L	print and keep
H	hold
<blank>	not specified

SUID=cccccccc

Selects a specific printout spool file with this sender(ID).

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "cccccccc" is the 1 - 8 byte alphanumeric name that is the Com-plete user ID.

LDRV=ccccccc

Selects a specific printout spool file with this output formatting routine.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "ccccccc" is the 1 - 8 byte alphanumeric name of the output formatting routine the user assigned to the printout spool file when it was created.

DEST=ccccccc

Selects a specific printout spool file with this logical printer name or terminal ID.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "ccccccc" is the 1 - 8 byte alphanumeric logical printer name to which the user sent the printout spool file when it was created.

DAT1=dd.mm.yy,mm/dd/yy

Selects a specific printout spool file with exactly this creation date.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "dd.mm.yy" (or "mm/dd/yy") is the 8 byte alphanumeric date assigned to the printout spool file when it was created.

DAT2=dd.mm.yy,mm/dd/yy

Selects a specific printout spool file with exactly this creation date.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "dd.mm.yy" (or "mm/dd/yy") is the 8 byte alphanumeric date assigned to the printout spool file when it was created.

If both date options are specified, they no longer describe these dates explicitly: they describe a date interval with DAT1 =/< DAT2.

TIM1=hh:mm

Selects a specific printout spool file with exactly this creation time.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "hh:mm" is the 5 byte time value assigned to the printout spool file when it was created.

TIM2=hh:mm

Selects a specific printout spool file with exactly this creation time.

If this option is omitted, the printout spool file data will be scanned for records meeting the other selection criteria specified.

Note that "hh:mm" is the 5 byte time value assigned to the printout spool file when it was created.

If both time options are specified, they no longer specify exact times: they describe a time interval with `TIM1 =/< TIM2`.

Examples

The following examples illustrate how the control statements can be used to select the desired options (printout spool files). To print a specific printout spool file, you may specify:

```
PRINT ,LNAM=NG1
```

To backup all printout spool files that were sent to printer ="Hugo" on February 29, 1988, specify:

```
B ,DEST=HUGO ,DAT1=29.02.88
```

or:

```
B ,DEST=HUGO ,DAT1=02/29/88
```

To restore all printout spool files that were sent from user ID ="Emma" in the period from January 4, 1988 to February 3, 1988, you may specify:

```
R ,SUID=EMMA ,DAT1=04.01.88 ,DAT2=03.02.88
```

You may also write the control statements in more than one line:

```
R ,
    SUID=EMMA ,
    DAT1=04.01.88 ,
DAT2=03.02.88 .
```

TUMSUTIL conventions

The following table summarizes the TUMSUTIL coding conventions.

Feature	Convention
SYSIN/SYSIPT	Control statements in the format: <code>command,option(s)</code>
PARM/SYSPARM	Format: "SCAN","NHDR"
	SCAN only perform syntax checking NHDR no header requested (PRINT request only)
User Exits	None
Return Codes	00 Normal completion 08 No core available, I/O error on data set

TUSACAPT - Capture File Initialization Utility

When the Capture files have first been allocated or they have been filled and emptied again, this utility must be used to reinitialize them for use. It simply allocates the number of specified records in the actual Capture data set(s) specified.

TUSACAPT is described under the following headings:

- How To Use TUSACAPT
- TUSACAPT conventions

How To Use TUSACAPT

The following example shows the job control required to initialize capture datasets.

```
//TUSACAPT JOB job-card information
//INIT      EXEC PGM=TUSACAPT
//STEPLIB   DD DSN=COM.LOAD,DISP=SHR
//CAPTURn   DD DSN=COM.CAPTURn,DISP=SHR

* $$ JOB JNM=TUSACAPT,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB TUSACAPT
// OPTION PARTDUMP
// LIBDEF PHASE,SEARCH=SAGLIB.COMvrs
// ASSGN SYS0001,DISK,VOL=.....,SHR      Assign for Capture dataset
// DLBL CAPTURn,'COM.CAPTURn'
// EXEC TUSACAPT
/*
/&
```

DD/DLBL	Description
CAPTURn	One or more Capture data sets to be initialised.

TUSACAPT conventions

The following table summarizes the TUSACAPT coding conventions.

Feature	Convention
User exits	None
Return Codes	00 Normal Completion

TUSDUTIL -SD File Maintenance Utility

The batch utility TUSDUTIL provides the following facilities:

- Initialization of VSAM COMSD dataset;
- Backup all SD files with optional deletion of SD files that have not been opened in the last *n* days;
- List contents of backup dataset;
- Restore SD files (selective or all) from backup dataset to COMSD data set.

TUSDUTIL is described under the following headings:

- Initialization of the Com-plete SD Data Set
 - Backup and Restoration of SD Files
-

Initialization of the Com-plete SD Data Set

Sample execution job control for allocation and initialization of this data set is given in the distribution Com-plete source library in member JCLINSTE. For more information on structure and allocation of the SD dataset, refer to **Com-plete Files and Associated User Files** and to **The Com-plete Task Structure** in this documentation.

The job control required to allocate and initialize a Com-plete SD data set is illustrated below:

MVS:

```
//TUSDINIT JOB    ... job-card information ...
//ALLOC      EXEC  PGM=IDCAMS
//SYSPRINT DD   SYSOUT=A
//SYSIN      DD   *
DELETE COM.SD
DEFINE CLUSTER -
    ( NAME (COM.SD) -
      NUMBERED -
      SHAREOPTIONS (2) -
      SPEED REUSE ) -
DATA -
    ( NAME (COM.SD.DATA) -
      CISZ (c) -
      RECORDSIZE (r r) -
      VOLUMES (vvvvvv) -
      CYLINDERS (ss))
/*
//INIT        EXEC  PGM=TUSDUTIL,
//            PARM='INIT,RECORDS=x,SDFILES=y,DMPSPAC=z'
/*
//STEPLIB DD     DSN=COM.LOAD,DISP=SHR
//COMSD DD      DSN=COM.SD,DISP=OLD
//SYSPRINT DD   SYSOUT=A
```

VSE:

```

* $$ JOB JNM=TUSDINIT,DISP=D,CLASS=? ..... POWER JOB CARD INFORMATION
* $$ LST DISP=D,CLASS=A
// JOB TUSDINIT ..... JOB CARD INFORMATION
// EXEC IDCAMS,SIZE=AUTO
/* DELETE (COM.SD) CLUSTER - */
/* CATALOG (catalog-file-name) */
/* */
DEFINE CLUSTER -
( NAME (COM.SD) -
NUMBERED -
SHAREOPTIONS (2) -
SPEED REUSE ) -
DATA -
( NAME (COM.SD.DATA) -
CISZ (c) -
RECORDSIZE (r r) -
VOLUMES (vvvvvv) -
CYLINDERS (ss)) -
CATALOG (catalog-file-name)
/*
// LIBDEF PHASE,SEARCH=SAGLIB.COMvrs,TEMP <---- Note 1
// DLBL COMSD,'COM.SD',,VSAM,CAT=COMCAT
// DLBL COMCAT,'catalog-file-name',,VSAM
/*
// EXEC TUSDUTIL,SIZE=AUTO,PARM='INIT,RECORDS=x,SDFILES=y,DMPSPAC=z'
/*
/&
* $$ EOJ

```

1. vrs in these cases relates to the Version, Release and SM level of the Com-plete being installed.

Parameters:

CISZ (c)	VSAM control interval size. This value influences utilization rate of disk space and the size of buffers allocated by Com-plete for access to the data set.
RECORDSIZE (r r)	VSAM record size (must be specified twice). Minimum value is 512. This is the "blocksize", into which Com-plete will block or split the logical records of all SD files. It should be as high as possible, must be greater than half CI size and must not exceed CISZ-12.
RECORDS=x	Number of VSAM records that are to be initialized. This parameter is useful only if you want to allocate a data set of more than 1 extent.Default: Maximum number of records that can be written to the extent(s) currently allocated for the data set.
SDFILES=y	Defines the size of the SD file directory (maximum number of SD files).Default: 500
DMPSPAC=z	Specifies the amount of space in Mbytes, that are to be assigned for thread dumps. The rest of the dataset will be used for SD files.Default: 50% of the dataset100% if SDFILES=0 is specified explicitly.

Backup and Restoration of SD Files

Contents of the Com-plete SD data set (including online dumps) can be backed up and restored using standard VSAM utilities. This is possible only when Com-plete is not active, and no selective backup and restoration of SD files is possible using these features.

Experience shows that, conditioned by the work file nature of most SD files, in time directory and SD file space of the data set become exhausted. This is mostly caused by application programs not deleting their work files when they become obsolete. To avoid abnormal program terminations caused by exhausted SD file directory or space, it is recommended periodically to run a BATCH backup job using the option to delete all SD files that have not been opened within a given number of days.

The job control required for backup and restoration of SD files is illustrated below:

MVS:

```
//TUSDUTIL JOB    ... job-card information ...
//              EXEC  PGM=TUSDUTIL,PARM='parm'
//STEPLIB DD      DSN=COM.USER.LOAD,DISP=SHR
//              DD      DSN=COM.LOAD,DISP=SHR
//              DD      DSN=ADABAS.LOAD,DISP=SHR
//BACKUP DD       DSN=COM.SD.BACKUP,DISP=OLD
//SYSPRINT DD     SYSOUT=A
//SYSIN DD        *
control card input for selective restoration of SD files
/*
```

VSE:

```
* $$ JOB JNM=TUSDUTIL,DISP=D,CLASS=? ..... POWER JOB CARD INFORMATION
* $$ LST DISP=D,CLASS=A
// JOB TUSDUTIL ..... JOB CARD INFORMATION
/*
/* =====
/* Com-plete MUST BE ACTIVE WITH ACCESS SUPPORT
/*
/* =====
/*
// LIBDEF PHASE,SEARCH=(SAGLIB.COMUSER, /* load ACSTAB */ *
SAGLIB.COMvrs, <---- Note 1 *
SAGLIB.CADAvrs),temp <---- Note 1
/*
/* =====
/* DLBL and EXTENT information for backup on DISK
/* =====
// DLBL BACKUP,'COM.SC.BACKUP',10,SD
// EXTENT SYS002,vvvvvv,1,0,ttttt,nnnn <---- Note 2
/*
/* =====
/* TLBL information for backup on TAPE
/* =====
/* TLBL BACKUP,'COM.SC.BACKUP'
/* ASSGN SYS002,cuu <---- Note 3
/*
/* =====
/* DLBL and EXTENT information for restore from DISK
/* =====
// DLBL RESTORE,'COM.SC.BACKUP',10,SD
```

```

// EXTENT      SYS001,vvvvvv,1,0,tttttt,nnnn          <----- Note 2
/*
/* =====
/* TLBL information for restore from TAPE
/* =====
/* TLBL        RESTORE,'COM.SC.BACKUP'
// ASSGN       SYS001,cuu                             <----- Note 3
/*
// EXEC        TUSDUTIL,SIZE=AUTO,PARM='parm'
control card input for selective restoration of SD files
/*
/&
* $$ EOJ

```

1. vrs in these cases relates to the Version, Release and SM level of the Com-plete being installed.
2. Change vvvvvv to the required VOLSER; change tttttt to the required TRACK and change nnnn to the required NUMBER of TRACKS.
3. Change cuu to the required TAPE unit address.

Parameter Values:

BACKUP	All SD files are written to the sequential data set defined by the DD / DLBL or TLBL statement BACKUP.
,DELAGE=n	(Allowed only if BACKUP is specified as first parameter) All SD files that have not been opened in the last n days are deleted from COMSD data set after they have been written to BACKUP.
RESTORE	SD files are restored from the sequential data set defined by the DD / DLBL or TLBL statement BACKUP. If a SYSIN DD (SYSIPT) statement is specified and SYSIN is not empty, selective restore will be performed.Default: All SD files are restored. Note: SD files existing in the COMSD data set are never overridden.
LIST	Contents of the dataset defined by DD / DLBL or TLBL statement BACKUP are listed on SYSPRIN/SYSLIST.

DD / DLBL or TLBL Statements:

BACKUP	With PARM option BACKUP: Sequential output data set to which the SD files are copied.Default DCB parameters: RECFM=VB,BLKSIZE=15000
RESTORE	With PARM option RESTORE / LIST:Sequential data set containing the unloaded SD files written by TUSDUTIL or by the TUSRSDCM utility of previous versions of Com-plete.
SYSPRINT/SYSLIST	Listing file
SYSIN/SYSIPT	Contains control cards for selective restoration of SD files.The format of the control cards is as follows:
	Columns 1-8 The SD file name (last two bytes must be blanks in Com-plete 4.5).
	Columns 9-13 The TID number of the SD file. Must be entered as a five-digit number with leading zeros, if necessary. If the SD file was created with SHR, then 'SHR' should be entered beginning in column nine.
	If more than one SD file is being selectively restored, one control statement must exist for each SD file. In addition, the control statements must be arranged in sequential, ascending order, by file name and TID number.

Sample execution job control for SD file backup and restoration is given in the distribution Com-plete source library in member JCLTUSDU.

Return Codes

0	Function executed successfully.
4	Check output for further error information
16	Parameter error.

If your job contains a SYSPRINT DD statement, all messages recieved from Com-plete will be logged to SYSPRINT.

Note:

Com-plete must be active with sysparm SUBSYS-ACTIVATE=ACCESS and defined in ACSTAB when this utility is used with PARM options BACKUP or RESTORE.

For detailed information on ACCESS, refer to **Software Interfaces** and to the Com-plete Installation and Migration documentation.

Miscellaneous Tables and Control Blocks

This part of the Com-plete System Programming documentation presents tables and control blocks for your reference.

This information is organized under the following headings:

- ULOG Info Table
- ULSODDT1 - SYSPRINT Routing Table
- ULPGMTAB - The Permanent Program Table
- Terminal Device Type Codes
- UED Edit Control Block
- UED Pseudo-Open Control Block
- UEDTB1 Entry DSECT
- UPDTB1 Information Control Block
- 3101 Terminal Support
- VTAM Logmodes

ULOG Info Table

UITAREA	DS	OD	
UITCODE	DC	H'0'	Function Code; corresponds to a "ULGnnnn" message, where nnnn=function code (See the Com-plete Messages and Codes documentation.)
UITCANC	DC	C' '	Reserved
UITWRM*	DC	C' '	Write logon screen: N - no map wanted
UITPWDR*	DC	C' '	Password required?: N - no
UITNBROD*	DC	C' '	Broadcast message: non-blank - no broadcast
UITUID	DC	CL8' '	Entered User ID
UITPWD	DC	CL12' '	Entered password
UITNPWD	DC	CL12' '	Entered new password
UITGRP	DC	CL8' '	Entered groupname
UITCMOD*	DC	CL8' '	Com-plete User ID model
UITTID	DC	H'0'	Com-plete TID number
UITACC*	DC	CL12' '	Account number, from User ID block
UITAC*	DC	H'0'	Authorization code
UITCTRL*	DC	C' '	Control status: C - control N - non-control
UITPRTY*	DC	C' '	TIB priority; possible values: C'0' to C'3'
UITUSER*	DC	F'0'	User field; used by ULOGX1 and ULSRPSFS
UITNEX1	DC	H'0'	Number of times ULOGX1 was entered
UITPVNUM	DC	H'0'	Number of password violations since Com-plete startup time
UITNAT*	DC	X'00'	Special flag
UITNATN	EQU	X'80'	X'80' - NATURAL UID
UITNATC	EQU DC	X'40' X'00'	X'40' - NO Com-plete UID Available
UITSMC*	DC	X'00'	Send Message classes
UITRMC*	DC	X'00'	Receive Message classes Rightmost bit class 1 Leftmost bit class 8

UITMSG*	DC	CL80' '	Error message text; contains a "ULGnnnn" message (See the Com-plete Messages and Codes documentation.)
UITMAPNM*	DC	CL6' '	Map name for ULOG
UITLCAS*	DC	C' '	Upper/lower case ' ' - no modification L - lower case U - upper case
UITHC*	DC	CL8' '	Hard copy device
UITNDBID*	DC	X'00'	NATURAL SECURITY File DBID
UITNFNR*	DC	X'00'	NATURAL SECURITY File FNR
UITPASS	DC	A(0)	Address of CCPASS describing optional ULOG PASS parameters.
UITL	EQU	*-UITAREA	Length of CUIT

The Com-plete ULOG Info Table (CCUIT) is created by the ULOG utility program before control is passed to the user-written exit routine ULOGX1. The information in this control block is used to initialize the user ID accounting block.

The ULOGX1 user-written exit may modify any of the fields marked with an "*" below in this control block.

ULSODDT1 - SYSPRINT Routing Table

This chapter describes the SYSPRINT capture table, which defines the DD/DLBL names for sequential SYSOUT type files.

This chapter covers the following topics:

- Overview
 - How to Code ULSODDT1
-

Overview

ULSODDT1 is a customizable table of DD names of SYSOUT (MVS) or SYSLST (VSE)-type files.

The purpose of this table is to provide a means of separating output data streams of different programs and / or different users directed to the same DD name, and to re-direct these output streams.

For each of the DD names specified in ULSODDT1, Com-plete internally converts every OPEN-PUT-CLOSE sequence into a PSOPEN-PSPUT-PSCLOSE sequence, creating a separate printout in Com-plete's printout spool.

It is possible to define a fixed destination (a printer or a SYSOUT class) for each of the DD names in ULSODDT1.

If, for any given DD name, no destination is defined in ULSODDT1, then the terminal user's hardcopy printer will be used. If the user has no hardcopy printer defined, then the output stream will be directed to the user's terminal, being displayed when the CLEAR key is pressed.

Note that some programs may test for the existence of the DD name before attempting an OPEN.

In this case, it may be necessary to add the appropriate DD statements to the Com-plete JCL.

How to Code ULSODDT1

ULSODDT1 is coded as an Assembler language module. It consists of a header and one line per DD name. Each of these lines consists of the 8-byte DD name, followed by the 8-byte destination. If you

Specify a non-blank destination, the output streams of all users for this DD name will be routed to this destination, no matter what the users' hardcopy destinations are.

Software AG recommends that you code your ULSODDT1 exactly as shown in the example or in the sample member provided in the Com-plete source library.

You can change the DD names and the destinations, add or remove lines between the lines labeled FIRST and FREE.

Example:

```

ULSODDT1  CSECT      ,
ULSODDT1  AMODE     31
ULSODDT1  RMODE     ANY
           DC        CL8 'ULSODDT1 '
           DC        A((FREE-FIRST)/16)          NUMBER OF DDNAMES
           DC        A(0)
FIRST     DC        CL8 'SYSOUT  ',CL8' '
           DC        CL8 'SYSPRINT',CL8' '
           DC        CL8 'SYSDBOUT',CL8' '
           DC        CL8 'CEEDUMP ',CL8 'CEEDUMP '   FOR LE/370
           DC        CL8 'SOMEDDN ',CL8 'SYSOUT=X'
FREE      DC        CL8' ',CL8' '          ZAP ROOM
           END
    
```

Note:

The format of ULSODDT1 used in previous versions of Com-plete is also supported further.

ULPGMTAB - The Permanent Program Table

In many systems, there are programs which must always be active, as they monitor some critical part of the system. In a TP system, it is sometimes difficult to ensure that a program stays permanently active. Com-plete provides a mechanism that ensures that a particular user program remains active and if it is not, Com-plete will reattach it as a user.

This chapter covers the following topics:

- Functional Overview
 - Building ULPGMTAB
 - Activating ULPGMTAB
-

Functional Overview

The application program must issue an OS ENQ for a particular major and minor name as soon as it is started. This ENQ must then be identified in the permanent program table, and Com-plete will check to see if this ENQ is available every 60 seconds. If the ENQ is available, it indicates that the program is not active and therefore Com-plete will restart the program whose name is associated with the ENQ in the table. Note that it is the user's responsibility to ensure that the ENQs are unique within the system and that the correct program uses the correct ENQ. In general, ENQs with a major name of COMPLETE and/or ADATPFE are reserved for internal use.

Building ULPGMTAB

The table is built using the CMPPGMTAB macro. The syntax is as follows:

```
CMPPGMT <function>,
PGM=<program>,
ENQARG=<enq list form expansion>
```

where:

<function>

Possible options:

GENTAB	Starts the generation of a table. This must be the first CMPPGMT entry in the program.
TABEND	Ends the generation of a table. This must be the last CMPPGMT entry in the program. ' ' : for an entry in the table.

<program>

is only valid for an entry in the table and is the name of the program to be started should the ENQ be found to be available.

<Enq list...>

is only valid for an entry in the table and is the address of the list form of the ENQ macro which is the ENQ that will be used to test whether or not the applicable program is active. This expansion must be generated as within the permanent program table module.

The ULPGMTAB supplied on the Com-plete distribution source data set is an example which shows how the Com-plete UTIMRM utility is defined.

Activating ULPGMTAB

To cause the permanent program table to be activated, it must be defined as a resident program. The module can have any CSECT name, the module name and therefore the resident program name must be ULPGMTAB.

Terminal Device Type Codes

The terminal device type codes listed in the following figure are used exclusively in the creation of TIBTAB. The columns abbreviated SS, LL, and LD represent:

Default SS	Screen Size, in characters
Default LL	Line Length
Default LD	Line Depth (number of lines)

The values listed for these items can be changed within the TIBTAB by specifying the keyword arguments FORMAT, LEN, and LINES when coding the TIB.

Terminal	Terminal Type	SS	LL	LD
Device Code				
BATCH	(Indicates that TIB may be used by batch)			
TTY	TELETYPE	-	80	26
TTYD	TELETYPE (DIAL)	-	80	26
3270 L	IBM 3270 LOCAL	1920	80	24
3270 R	IBM 3270 REMOTE	1920	80	24
3275 R	IBM 3275 REMOTE	1920	80	24
3278 L	IBM 3278 LOCAL	1920	80	24
3278 R	IBM 3278 REMOTE	1920	80	24
3279 L	IBM 3279 LOCAL	2560	80	32
3279 R	IBM 3279 REMOTE	2560	80	32
3284 L	IBM 3284 LOCAL	-	120	20
3284 R	IBM 3284 REMOTE	-	120	20
3286 L	IBM 3286 LOCAL	-	120	20
3286 R	IBM 3286 REMOTE	-	120	20
3287 L	IBM 3287 LOCAL	-	132	18
3287 R	IBM 3287 REMOTE	-	132	18
3288 L	IBM 3288 LOCAL	-	132	18
3288 R	IBM 3288 REMOTE	-	132	18

UED Edit Control Block

The UED edit control block is addressed by the parameter list passed to the user-written exit UUEDEX. The portion of the edit control block listed here is a partial listing and includes that portion of the edit control block preceding the GETCHR information area.

Location		Length	Format	Contents
Dec	Hex			
0	0	16	Binary	Com-plete registers 2 through 5.
16	10	4	Binary	Address of the command table.
20	14	4	Binary	Current command table entry.
24	18	4	Binary	Address of the command buffer.
28	1C	4	Binary	Buffer address for the communication line.
32	20	4	Binary	Buffer address for the repeat command.
36	24	4	Binary	Address of the current CMD/operand.
40	28	4	Binary	Address of the next command (or 0).
44	2C	4	Binary	Address of the next operand (or 0).
48	30	4	Binary	Address of the output buffer.
52	34	4	Binary	Current position in the output buffer.
56	38	4	Binary	Address of the SD buffer.
60	3C	4	Binary	Address of the first macro buffer.
64	40	4	Binary	Address of the current macro processor save area.
68	44	4	Binary	Address of the 'HALT X' exit routine.

Additional entries in the UED edit control block do exist. They can be obtained by referring to the expansion of the CMEDTB1 macro.

The terminal device type codes listed in the following figure are used exclusively in the creation of TIBTAB. The columns abbreviated SS, LL, and LD represent:

SS	Screen Size, in characters
LL	Line Length
LD	Line Depth (number of lines)

The values listed for these items may be changed within the TIBTAB by specifications of the keyword arguments FORMAT, LEN, and LINES when coding the TIB and LGCB macros.

Terminal	Terminal Type	SS	LL	LD
Device Code				
BATCH	(Indicates that TIB may be used by batch)			
TTY	TELETYPE	-	80	26
TTYD	TELETYPE (DIAL)	-	80	26
3270 L	IBM 3270 LOCAL	1920	80	24
3270 R	IBM 3270 REMOTE	1920	80	24
3275 R	IBM 3275 REMOTE	1920	80	24
3279 L	IBM 3279 LOCAL	2560	80	32
3279 R	IBM 3279 REMOTE	2560	80	32
3284 L	IBM 3284 LOCAL	-	120	20
3284 R	IBM 3284 REMOTE	-	120	20
3286 L	IBM 3286 LOCAL	-	120	20
3286 R	IBM 3286 REMOTE	-	120	20
3287 L	IBM 3287 LOCAL	-	132	18
3287 R	IBM 3287 REMOTE	-	132	18
3288 L	IBM 3288 LOCAL	-	132	18
3288 R	IBM 3288 REMOTE	-	132	18

UED Pseudo-Open Control Block

The UED pseudo-open control block is addressed by the parameter list passed to the user-written exit UUEDEX. The data in this control block is used to identify the file being edited and to provide additional information about that file.

Location		Length	Format	Contents
Dec	Hex			
0	0	44	Character	File name
44	2C	8	Character	Member name
52	34	6	Character	Volume serial number
58	3A	2	Character	Two-character library identification
60	3C	4	Binary	Address of I/O control blocks
64	40	4	Binary	DSCB address
68	44	4	Binary	DCB address
72	48	4	Binary	DEB address
76	4C	4	Binary	IOB address
80	50	4	Binary	UCB address
84	54	4	Binary	Buffer address
88	58	4	Binary	Address of library table entry
92	5C	2	Binary	DSORG indicator
94	5E	1	Binary 1..... .1..... ..1..... ...1....1...	Flag byte 1: Open for output Output, only shared ENQ Open completed Error call for U2EDXCLS Dev permanently resident
95	5F	1	Binary 1..... .1..... ..1..... ...1....1...1..	Flag byte 2: Freemain I/O control blocks Dequeue SYSDSN Dequeue SYSIEWLP Deallocate UCB Freemain format 3 DSCB Freemain buffer
96	60	2	Binary	Additional flags

UEDTB1 Entry DSECT

The UEDTB1 module is a table of two-character library codes and information about the associated libraries. At invocation time, the utility programs UED, UEDIT, USERV, and UPDS load module UEDTB1. Each of these utility programs passes to its respective user-written exit routine a pointer to an entry in this module, if a two-character library code was used when accessing a library.

The following DSECT summarizes the format of each entry.

Location		Length	Format	Contents
Dec	Hex			
0	0	2	Binary	Length of this entry (includes user data field)
2	2	2	Character	Library code
4	4	6	Character	Volume serial number, unless cataloged
10	A	2	Binary	Sequence number columns: First byte - Start column Second byte - End column
12	C	2	Binary	Tag number columns: First byte - Start column Second byte - End column
14	E	10	Binary	Paired bytes indicating start and end columns for the UED list function
24	18	10	Binary	UED tab stop columns
34	22	1	Binary	Access method: x'8x' - no user data display x'4x' - no user data stow x'2x' - input only x'x0' - LIBRARIAN x'x1' - Sequential x'x2' - PANVALET x'x3' - LIBRARIAN
35	23	1	Binary	DSN length, minus one
36	24	xx	Binary	User data field xx = maximum of 176
aa	bb	44	Character	File name aa = 36 + xx bb = aa in hexadecimal

UPDTB1 Information Control Block

The UPDTB1 information control block is used by UPDS and USERV in order to control access to each reference file. This control block contains information about the file being accessed and the command request being given. The address of this control block is passed to the user-written exit UUPDX1 as the second argument in the parameter list. This table is also described by the UPDTB1 macro. Assemble the sample UUPDX1 to see the layout of this area.

Note:

Fields not expressly noted in the following control block are reserved.

Location		Length	Format	Contents
Dec	Hex			
1608	648	44	Character	File name
1652	674	6	Character	Volume serial number
1658	67A	1	Character	DOS sub-library type
1659	67B	2	Character	Two-character library identification code, or spaces
1667	683	8	Character	Member name
1717	6B5	8	Character	Printout spool code
1725	6BD	8	Character	Default destination code
1735	6CD	8	Character	UPDS/USERV command
1756	6DC	1	Binary	DSN length, minus 1
1764	6E4	1	Binary	Flag 1: 1..... SYSDSN enqueued .1..... SYSIEWLP enqueued ..1..... UCB allocated ...1.... Only shared ENQ on SYSDSN1... UCB permanently resident1.. File is a PDS
1765	6E5	1	Binary	Open status: 1 - Open for input 2 - Open for output
1702	6A6	1	Binary	Flag 2: 1..... Reserved .00.... Character display .11.... HEX display .01.... Interpretive HEX ...1.... Member entered in command1... Formattable device1.. Buffer too small for block1. Position parm with no keyword1 Reserved
1703	6A7	1	Binary	Flag 3: 1..... User data option Y .1..... User data option X ..1..... Reserved ...1.... User data option T

3101 Terminal Support

The 3101 is supported by Com-plete either as a standard TTY device or in 3270-emulation mode. The mode is defined in Com-plete's terminal table. In emulation mode, the 3101 appears to the application program as a 3270. Com-plete translates the application program's 3270 data stream to 3101 format and then translate the 3101 data stream to 3270 format for the application program.

Note:

There are some device-dependent differences of which the application programmer should be aware. These differences are explained later in this documentation in the section **Programming Considerations**.

This chapter covers the following topics:

- Operation
 - Programming Considerations
 - System Programming Considerations
-

Operation

This section explains the use of the 3101 keyboard and the use of the 3101 setup switches. Since Com-plete begins the data stream with a LOCK ORDER and end it with an UNLOCK ORDER, a transmission error may result in a locked keyboard. This condition can be cleared by toggling the NORMAL/TEST key. It can also be cleared by use of the BREAK key. In 3270 emulation mode, however, the BREAK key will result in an emulated CLEAR key (end-of-job).

Keyboard Functions

SEND KEY	Is used as the equivalent to the 3270 ENTER key. In emulation mode, SEND performs the 3101 SEND-PAGE operation.
NEW LINE KEY	Moves the cursor to the next line, but does not TAB to the first field on that line as does the 3270 RETURN key.
CLEAR KEY	Causes the screen to be erased, but does not transfer data to the host. To emulate the 3270 CLEAR function, use the BREAK key or the CLEAR key; then type in *EOJ, and press the SEND key.
SEND MSG	Sends data up to the cursor location. This key can be used when the screen is not formatted.
SEND LINE	Sends one line of data. This key is most useful in TTY mode and should be avoided in emulation mode.
ERASE EOS KEY	Clears all unprotected data from the screen starting at the cursor location to the end of the screen.
ERASE EOF KEY	Clears all unprotected data from the cursor location to the end of the field. This key is the same as the 3270 ERASE EOF key.
ERASE INPUT KEY	Causes all unprotected data to be erased from the screen. This key is equivalent to the 3270 ERASE INPUT key.
TAB AND	These keys cause the cursor to move from one BACK-TAB KEYS adjacent field to another. They function the same way as the 3270 TAB keys. Unlike the 3270, the 3101 has static TAB locations in the first and last screen locations. In addition, the 3101 does not automatically skip to the next field when the current field fills with data; therefore, the TAB key must be used for each field.
PF KEYS	There are only eight PF keys on the 3101. They do not cause data to be transmitted to the host. Com-plete translates the 3101 PF keys as follows: a.PF1 through PF6 are translated as PF1 through PF6. b.PF7 and PF8 are translated as PA1 and PA2.

Setup Switches

Com-plete requires that some of the 3101 setup switches be set in a certain manner, while others are either optional or dependent on the hardware in use. The following table summarizes the setup switches. The switch name is preceded by (X,Y), where X is the group number of the switch and Y is the number of the switch within the group.

Switch	Description
(1,1) BLOCK/CHAR	Sets the 3101 transmission mode. Com-plete 3270 emulation requires the BLOCK (ON) setting.
(1,2) HDX/FD	Indicates half- or full-duplex. BLOCK assumes HDX (ON).
(1,3) MODEM TYPE	Setting is dependent on the modem type. The normal setting is (ON).

Switch	Description
(1,4) PRTS/CRTS	Controls the request to send status. This is normally set to (ON).
(1,5) REVERSE CH	Specifies whether or not a REVERSE CHANNEL ON/OFF is in use. It is normally set to (OFF).
(1,6/7) EOT/ETX/	Selects the line turnaround characters. CR/XOFF Com-plete requires that the setting be CR (ON,OFF).
(1,8) DUAL/MONO	Sets the character set mode. It is normally set to MONO (OFF).
(2,1) STOP1/STOP2	Sets the number of stop bits used in the line protocol. The number of bits is usually dependent on the line speed. This is normally set to STOP1 (ON).
(2,2/3) ODD/EVEN/	Controls the line protocol parity type. The MARK/SPACE Com-plete default is SPACE (OFF,OFF).
(2,4) SEND LINE	Is used to change the function of the SEND key from SEND-PAGE to SEND-LINE. Com-plete 3270 emulation requires that this key be set to SEND-PAGE (OFF).
(2,6) NULL SUPP	Controls the suppression of the transmission of the trailing NULL characters. Com-plete emulation requires NULL SUPP (ON).
(2,7/8) NO OF TIME	Controls the number of TIME-FILL characters FILL CHARS used for the print data stream. This should be set to (OFF,OFF).
(3,1) AUTO NL	When (ON), automatically moves the cursor to the ON/OFF first character position in the next line. This is assumed to be (ON) when in BLOCK mode.
(3,2) AUTO LF	Controls the positioning of the cursor when the CR ON/OFF character is received. Com-plete emulation requires the (ON) setting.
(3,3) CR/CR LF	Controls the characters generated by the NL key. It can generate CR, or CR and LF. This is not applicable in BLOCK mode, but is normally set to CR (ON).
(3,4) SCROLL ON/OFF	Controls the scrolling mode of the 3101. Com-plete emulation requires no scrolling, that is, SCROLL (OFF).
(3,7) REVERSE VIDEO	Controls the use of the reverse video display feature. This setting is optional.
(3,8) BLINK CURSOR	Makes the cursor blink. This setting is optional.
(4,1-4) LINE SPEED	Is set according to the speed of line in use. The following table contains the appropriate settings. (1=ON,0=OFF)

Switch	Description	
	BPS	Switches
	150	0001
	200	0010
	300	0011
	600	0100
	1200	0101
	1800	0110
	2400	0111
	4800	1000
(4,5-8)	Controls the transmission speed to the printer. See the above LINE SPEED table for the equivalent settings.	

Example

The following table contains a setup switch example. (1=ON,0=OFF)

1	2	3	4
11010010	10000100	11100000	01010101
BLOCK MODE	STOP1	CR.LF	1200 BPS
HDX	SPACE	SCROLL OFF	1200 BPS
CL/422	SEND PAGE	VIDEO	
PRTS	SUPP NULLS	NON-BLANK	
REV. CH. OFF	0 TIME-FILL		
CR			
MONO			

Programming Considerations

The 3101 appears to the Com-plete application programmer like a 3270 model 2. The following differences between the 3101 and the 3270 should be kept in mind, however.

- The cursor location is not returned by the 3101 SEND operation; therefore, the cursor address returned in the emulated 3270 buffer is always the highest screen location.
- The 3101 does not have a numeric attribute; therefore, the terminal operator is not limited to entering numeric characters in fields defined as numeric in the 3270 output.

- There is no equivalent to the 3270 ERASE-TO-ADDRESS order. Com-plete emulation ignores this order, unless the address in the order is the end of the screen. In that case, it generates the 3101 EOS function.
- The 3101 does not allow a "wraparound" field. In other words, it does not allow an attribute byte in the last screen position to define a field that begins in the first screen position. If Com-plete emulation encounters a situation where data wraps the screen, it will copy the last attribute byte encountered into the first screen location, thereby destroying the first screen location byte.
- The 3101 does not have the equivalent of the 3270 PT order. Com-plete emulation replaces the PT with the 3101 HT, which does not perform the ERASE EOF function before skipping to the next field. Because of this limitation, the Com-plete full-screen editor UEDIT will blank-fill fields instead of performing the PT.
- The Com-plete GETCHR function returns a device type of 327E. The PF Keys on the 3101 do not transmit data. Therefore, programs dependent on this 3270 feature will not work on the 3101. See the section entitled **Keyboard Functions** earlier in this documentation for PF emulation.
- The only WCC functions emulated by Com-plete are the SOUND ALARM and RESET KEYBOARD functions.

System Programming Considerations

In order to emulate a 3270 with the 3101, you must define the 3101 as a TTY device and specify OPT=EM3270. Other TIB parameters are coded just as they would be for normal TTY devices.

VTAM Logmodes

This chapter discusses some MODEENT parameters relevant for Com-plete to set up correct TIB information for the various partner LU types. The LOGMODE specifications override any static TIBTAB definitions.

This chapter covers the following topics:

- MODEENT Parameters
 - Sample LOGMODE Specifications
-

MODEENT Parameters

The format of a LOGMODE specification is:

```
name MODEENT LOGMODE=modename *
  TYPE= type, *
  FMPROF=fmprof,   Function Management Profile*
  TSPROF=tsprof,   Transport Services Profile*
  COMPROT=comprot, Common protocol *
  PRIPROT=priprot, PLU protocol*
  SECPROT=secprot, SLU protocol*
  RUSIZES=rusizes, Rusize=1K*
  PSERVIC=pservic
```

where:

type	must be (negotiable) for LU6.2 sessions and 1 for all other types (default).
fmprof (1)	X'03' for SNA terminals/printers X'02' for non-SNA terminals printers X'13' for LU6.2 sessions
tsprof (1)	X'03' for SNA terminals/printers X'02' for non-SNA terminals/printers X'07' for LU6.2 sessions
comprot	2-Byte hexadecimal value
Byte 1	<p>1000 0000 the sending node does not accept receipt of segments (reserved for nonextended non-LU 6.2)</p> <p>0100 0000 FM Headers allowed (only value defined for LU 6.2)</p> <p>00x0 0000 - 1 Brackets are used and reset state is BETB (in brackets) This bit should be 0 for LU6.2 sessions (FM PROF=X'13')</p> <p>000x 0000 - 1 Conditional Bracket termination (only value defined for LU6.2)</p> <p>0 Unconditional termination</p>
Byte 2	xx00 0000 - Normal-flow send/receive mode selection: 00 - full-duplex 01 - half-duplex contention 10 - half-duplex flip-flop (only value defined for LU6.2) 0010 0000 - Symmetric responsibility for recovery (only value for LU6.2) 0001 0000 - Primary is contention winner, secondary is contention loser 0000 0010 - Control vectors are included after SLU name 0000 0001 - HDX-FF reset state is SEND for primary and RECEIVE for secondary.
priprot	1000 0000 - multiple-RU chains allowed (only value defined for LU6.2) 0x00 0000 - 1= delayed request mode 0= immediate request mode (only value for LU6.2) 0000 xx00 - Chain response protocol used by primary LU: 00 = no response 01 = exception response 10 = definite response 11 = definite or exception response (only value for LU6.2) 0000 000x - 1= primary may send EB (End Bracket) 0= primary may not send EB (only value for LU6.2)
secprot	bits have the same meaning as for priprot but applicable to SLU
rusizes	Maximum RU sizes sent by primary and secondary half sessions. If bit 0 is set to 1 (only value defined for LU6.2) the Byte is interpreted as X'ab' = a*2**b.
pservic	is a 12 Byte Hexadecimal field returned to the application in the INQUIRE macro: Byte 0 : LU-Type (0, 1, 2, 3 and 6 are supported).

For LU6.2:

Bytes 1-8	PS Usage Field. the only value defined is X'0200000000000000'
Byte 9	Must be X'00' for Transaction Routing Sessions and SNASVCMG. Access security will not be accepted on incoming FMH-5s. X'10' for native user Sessions: Access security will be accepted on incoming FMH-5s. Already-Verified will not be accepted.
Byte 10	Should be X'2F': - Synclevel=CONFIRM - either PLU or SLU may reinitiate sessions - parallel sessions are supported - CNOS GDS variables are supported

For LU-types 2 and 3:

Byte 1	bit 0 (X'80') indicates this is a Queriable Device . TIBSQRY is set in the TIB entry and Com-plete will issue a READ-PARTITION-QUERY just after OPNDST to check support and values for: extended highlighting color load program symbol
Bytes 2 to 5	Reserved X'00000000'
Byte 6	Default number of lines(2)
Byte 7	Default line length(2)
Byte 8	Aternate number of lines(2)
Byte 9	Alternate line length (2)
Byte 10	Screen size: Screen sizes can be specified in Byte 10 or Byte 10 can refer to bytes 6 to 9. If the base of SNA data streams is used, byte 10 may specify only X'00', X'01' or X'02'. X'00' permits either 480 or 1920 screen size. If X'03' is specified (LU Type 2 only), the default screen size is 1920 and the alternate size is specified in the Query Reply structured field. If X'7E' is specified, the screen size specified (bytes 6 and 7) is static. When in this mode, an Erase/Write Alternate command may be rejected with sense X'1003'. If X'7F' (dynamic screen size switching) is specified, an alternate screen size may be specified in byte 8 and 9.
Byte 11:	Reserved (X'00')

For LU Type 1:

Byte 1	FM Header set
Bytes 2-6	Primary half session usage
Bytes 2 and 3	FM Header flags
Byte 4	Data Stream Flags:

Notes:

1. Some devices as gateways may require different values. Please refer to the device documentation.
2. Default and alternate screen sizes may be overwritten as result of the READ-PARTITION-QUERY.
3. Refer to the IBM documentation *SNA - Sessions Between Logical Units* for more information.

Sample LOGMODE Specifications

```

*
*LOGMODE for 3278 Model 5 Terminal
*
M3278S5 MODEENT LOGMODE=M3278S5,          *
  FMPROF=X'03',  Function Management Profile *
  TSPROF=X'03',  Transport Services Profile *
  COMPROT=X'3080', Common protocol          *
  PRIPROT=X'B1',  PLU protocol              *
  SECPROT=X'90',  SLU protocol              *
  RUSIZES=X'8787',          *
  PSERVIC=X'02800000000018501B847F00'
0 1 2 3 4 5 6 7 8 9 10 11
*
*          LOGMODE for SCS Printer
*
SCS  MODEENT LOGMODE=SCS,          *
  FMPROF=X'03',  Function Management Profile *
  TSPROF=X'03',  Transport Services Profile *
  COMPROT=X'3080', Common protocol          *
  PRIPROT=X'B1',  PLU protocol              *
  SECPROT=X'90',  SLU protocol              *
  RUSIZES=X'8787',          *
  SRCVPAC=1,     SLU: Receive Pacing Count *
  PSNDPAC=1,     PLU: Send Pacing Count    *
  PSERVIC=X'01000000E100000000000000'
0 1 2 3 4 5 6 7 8 9 10 11
*
* LOGMODE for SCS Printer on SAA Gateway
*
SCSSAA MODEENT LOGMODE=SCSSAA,          *
  FMPROF=X'03',  Function Management Profile *
  TSPROF=X'03',  Transport Services Profile *
  COMPROT=X'3080', Common protocol          *

  PRIPROT=X'B1',  PLU protocol              *
  SECPROT=X'90',  SLU protocol              *
  RUSIZES=X'8787',          *
  SRCVPAC=1,     SLU: Receive Pacing Count *
  PSNDPAC=1,     PLU: Send Pacing Count    *
  PSERVIC=X'0140000100000000001000000'
0 1 2 3 4 5 6 7 8 9 10 11
*
* LOGMODE for SNA Printer
*
DSC2K MODEENT LOGMODE=DSC2K,          *
  TYPE=1,  Nonnegotiable Bind              *
  FMPROF=X'00',  Function Management Profile *
  TSPROF=X'07',  Transport Services Profile *
  COMPROT=X'3080', Common protocol          *
  PRIPROT=X'B1',  PLU protocol              *
  SECPROT=X'90',  SLU protocol              *

```

```

RUSIZES=X'8787',          *
PSERVIC=X'030000000000185018507F00'
  0  1  2  3  4  5  6  7  8  9 10 11
*
*LOGMODE for SNA Service Manager Sessions - Mandatory for LU6.2 parallel sessions
*
SNASVCMG MODEENT LOGMODE=SNASVCMG, *
TYPE=0,      Negotiable Bind      *
FMPROF=X'13',  Function Management Profile *
TSPROF=X'07',  Transport Services Profile *
COMPROT=X'50B1', Common protocol      *

PRIPROT=X'B0',  PLU protocol      *
SECPROT=X'B0',  SLU protocol      *
RUSIZES=X'8787',  Rusize=1K      *
SRCVPAC=0,      SLU: Receive Pacing Count *
SSNDPAC=0,      SLU: Send Pacing Count *
ENCR=0,        No encryption,      *
PSERVIC=X'060200000000000000002F00'
  0  1  2  3  4  5  6  7  8  9 10 11
*
* LOGMODE for LU6.2 parallel sessions (no security)
*
LU62P MODEENT LOGMODE=LU62P,      *
TYPE=0,      Negotiable Bind      *
FMPROF=X'13',  Function Management Profile *
TSPROF=X'07',  Transport Services Profile *
COMPROT=X'50B1', Common protocol      *
PRIPROT=X'B0',  PLU protocol      *
SECPROT=X'B0',  SLU protocol      *
RUSIZES=X'8787',  Rusize=1K      *
SRCVPAC=0,      SLU: Receive Pacing Count *
SSNDPAC=0,      SLU: Send Pacing Count *
ENCR=0,        No encryption,      *
PSERVIC=X'060200000000000000002F00'
  0  1  2  3  4  5  6  7  8  9 10 11
*
* LOGMODE for LU6.2 parallel sessions (Security)
*
LU62PS MODEENT LOGMODE=LU62PS,      *
TYPE=0,      Negotiable Bind      *
FMPROF=X'13',  Function Management Profile *
TSPROF=X'07',  Transport Services Profile *
COMPROT=X'50B1', Common protocol      *
PRIPROT=X'B0',  PLU protocol      *
SECPROT=X'B0',  SLU protocol      *
RUSIZES=X'8787',          *
RCVPAC=0,      SLU: Receive Pacing Count *
SSNDPAC=0,      SLU: Send Pacing Count *
ENCR=0,        No encryption,      *
PSERVIC=X'060200000000000000102F00'
  0  1  2  3  4  5  6  7  8  9 10 11

```