

CA Telon[®] Application Generator

Programming Concepts Guide

r5.1



This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") are for your informational purposes only and are subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be used or disclosed by you except as may be permitted in a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2010 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Product References

This document references the following CA products:

- CA Telon® Application Generator (CA Telon)
- CA Panvalet®
- CA Endeavor Software Change Manager

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At <http://ca.com/support>, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, complete our short [customer survey](#), which is also available on the CA Support website, found at <http://ca.com/docs>.

Contents

Chapter 1: Introduction to Programming Concepts 19

Introduction	19
CA Telon	19
A Productivity Tool	20
A Tool that Meets Your Quality Requirements	20
A CASE Tool	20
A Control Tool	20
Features and Benefits	21
Analysis Tool Interface	21
Screen/Report Painting	21
Prototyping	21
Data Dictionary Interface	21
File Generation/Groupings	22
Generated DBMS Access	22
Design Capture	22
Flexibility	22
Environment Conversion	22
CA Telon Components and Program Development	23
CA Telon Design Facility (TDF)	23
CA Telon Application Generator	26
Online Program Architecture	28
Batch Program Architecture	28
Program Design Steps	29
CA Telon Specifications	29
Environments	29
Databases	30
Languages	31
Interfaces	31

Chapter 2: Designing CA Telon Programs 33

Designing with CA Telon Online	33
Screen Concepts	34
Screen Flow and Documentation	38
Screen Layout	44
Screen Map/Edit Chart	44
Screen Narrative	47
Design-Time Prototyping	48

Evolutionary Development	49
Designing with CA Telon Batch	52
Systems with Reports	52
Systems without Printed Reports	55

Chapter 3: CA Telon Design Facility **61**

TDF Program Structure	61
Final CA Telon Program	63
Programming with the TDF	66
Online TDF Program Structure	66
Creating a Panel Image	66
Creating a Panel Definition	68
Creating a Program Definition	69
Batch TDF Program Structure	71
Creating a Panel Image	71
Batch Report Events	73
Creating a Panel Definition	74
Creating a Program Definition	76
TDF Screen Organization	77
TDF Main Menu Screen Organization	77
TDF Option 1 Screen Organization	78
TDF Option 2 Screen Organization	79
TDF Option 3 Screen Organization	81
TDF Option 4 Screen Organization	83
TDF Option 5 Screen Organization	93
TDF Option 6 Screen Organization	96
TDF Option U Screen Organization	97
TDF Main Menu	98
Parameters	100

Chapter 4: User Profile Maintenance **105**

Organization of the User Profile Maintenance Screens	105
User Profile Maintenance Menu	105
Program Definition Defaults Example	106
Tasks	106
Update Program Definition Defaults	107
Update PF Key Definitions	109
Update Session Controls	111

Chapter 5: Creating a Panel Image **113**

TDF Main Menu	113

Creating an Online Panel Image	113
Field Types	115
Creating a Batch Panel Image	118
Factors Unique to Batch Panels	119
Batch Report Events	119
Field Types	123
Logical Groups	125

Chapter 6: Creating the Panel Definition 129

Panel Definition Structure	129
Organization of Panel Definition Screens	131
Creating an Online Panel Definition	132
Specify Field Input	134
Specify Further Updates	135
Define Consistency Edits	141
Creating a Batch Panel Definition	145
Specify Logical Groups	146
Update Logical Groups	148
Specify DBNames	150
Update Output Fields	151

Chapter 7: Program Hierarchical Structure 157

Online Program Hierarchical Charts	158
Program Overview	158
Main Processing	160
Output Processing	162
Input Processing	166
Batch Program Hierarchical Charts	173
Program Overview	173
Batch Mainline	174
Mainline Sort	179
Batch Mainline Sort Program Structure With I/O Proc	181
Batch Mainline Sort Program Structure (no input PROC)	182
Batch Mainline Sort Program Structure (no output PROC)	182
Batch Mainline Sort Program Structure (no PROCs)	183
User-Defined Sort	183
Generated Sort Objects	183
Batch Match	184
Batch Merge	187
Generated Merge Objects	188
Merge Processing	189

Program Control	190
CICS Nonterminal Program Structure	190
CICS Client Program Hierarchical Charts	194
Program Overview	194
CONTROL-INDICATOR	194
Main Processing	195
MAIN-LINE	196
MAIN-PROCESS	197
Output Processing	198
K-100-HOLD-RESTORE	198
C-500-FORM-INIT	198
B-100-OUTPUT-EDITS	199
SET 'DO-WRITE'	200
Input Processing	200
P-100-PFKEYS	201
C-600-FORM-PROCESS	202
SET 'DO-TRANSFER'	204
CICS Server Program Hierarchical Charts	204
Program Overview	204
CONTROL-INDICATOR	204
Main Processing	205
IMS-PRIMARY-ENTRY	205
MAIN-LINE	205
MAIN-PROCESS	206
Form Initialization Processing	207
A-100-OUTPUT-INIT	208
B-100-OUTPUT-EDITS	209
SET 'DO-WRITE'	210
PROCESS-FORM Processing	210
P-100-PFKEYS	212
D-100-INPUT-INIT	212
J-100-SELECT	212
PROCESS-FIELD Processing	215
E-200-PROCESS-FIELD	216
SET 'DO-WRITE'	216
TERM-FORM Processing	216
S-100-SERVER-TERM	217
Stored Procedure Hierarchical Charts	217
Program Overview	218
Open Files	219
Initialize	219
Process	219

Terminate	219
Close Files	220
Other Sections	220

Chapter 8: Custom Code **221**

Basics of Using Custom Code	221
Parameters to Request Custom Code	224
Custom Code Points for CICS Client	229
Edits	230
Field Edits	230
Consistency Edits	236
Examples of Consistency Checks	244
Attribute Considerations	247
Cursor Positioning	248
Standard Attributes	250
Overriding Attributes	250
Extended Attributes	250
Other Extended Attributes	251
Error Handling	251

Chapter 9: Processing Flow **253**

Controlling Processing Flow	253
NEXTPGM Parameter	254
CONSYS Parameter	254
SCONSIS Parameter	255
PFKEY Parameter	255
List Processing	256
SEGLOOP Processing	256
PF Keys	274
To Control Screen Flow	274
HOLD Processing	278
HELP Processing	280
SELECT Fields	281
Determining Where to Pass Control	283
Processing Considerations with SELECT Logic	283
Flow with List Screens	286
HELP/HOLD Processing	288
HOLD Requirements	289
HELP Requirements	293
HELP Message	295
PFKEY Use	298

IMS/DC Report Processing	301
Capabilities	301
Uses	302
Differences between Screens and Reports	302
Report Definition	302
Report Program Structure	304
Controlling the Report Destination	305
Calling Program Interface	306
Line Optimization Considerations	307
LINEOPT Parameter	307
OUTIFIL Parameter	308
EOFKEY Parameter	308
Refresh Considerations	309
Output Field Attributes	309

Chapter 10: Prototyping Facility 311

Prototyping without Data Mapping	313
Prototyping with Data Mapping	313
Prototyping Facility Menu	315
Position of Command Field	316
Initiating a Scenario from a List Screen	317
Prototype Screens	317
Command Field	318
Flow Control	318
Prototyping Commands	319
General TDF Commands	320
Methods to Control Screen Flow	320
Command Field Entry	320
.COMMAND-FLOW	321
NEXTPGM Parm of SELECT Fields in the PD	321
NEXT-PROGRAM-NAME-ID	321
NEXTPGM Parameter of Screen Definition	322
Terminating a Scenario	322
Modifying Application Definitions	322
Presentation Stores	323
Mapping from a Presentation Store	323
Contents of an Active Presentation Store	324
Modifying an Active Presentation Store	325
Saving and Retrieving a Presentation Store	326
Presentation Store Editor	327
Canned Demonstrations	328
Listing Presentation Stores	328

Prototyping List Screens	329
Handling Arrays from Non-List Screens	330
Subscripted and Unsubscripted Variables	330
Subscripted Variable Display Rules	331
Simulating Error Processing	332
Customizing Error Messages	332
Creating Canned Scenarios	333
Special CA Telon Field Names	333
Simulating Edit/Flow Scenarios	335
Input Mapping Considerations	336
Suggested Prototyping Steps	338
Displaying Screens with Sample Data	339
Realistic Scenarios	339
Control Hints	340
Sample Prototyping Sessions	340
Prototyping without Data Mapping	341
Prototyping Facility Menu	345
Prototyped Panel Images	346
Prototyping with Data Mapping	352
LINE Commands	354

Chapter 11: Creating the Program Definition 359

Create the Screen Definition	359
Specify Custom Code Editing	361
Create a Data Group	366
Preview Generated Data	368
Create the Batch Definition	370
Review the Panel Definition	375
Create a Data Group	377
Enter Custom Code	385
Specify Environment	390

Chapter 12: Initializing Applications 393

IMS/DC	393
Task 1: Define CA Telon Application COPY Members	393
Task 2: Define Database Segment COPY Members	400
Task 3: Create a TSO Test PSB	400
Task 4: Allocate Test Databases	400
CICS	401
hhWKAREA	401
hhUPDTA	403

File COPY Members	403
Transfer Work Area Definition	403
Information Included in the Transfer Work Area	404
Required Transfer Work Area Items	405
SELECT Key-Save Value	407

Chapter 13: Application System Generator 409

Components of a Definition	410
Data Search Criteria	410
Data Sorting, Matching, or Merging	410
Definition Statements	411
Data Access	412
Definition Literals	414
Definition Variables	414
Consistency Edits	415
Environment Statements	416
Statement Parameter Lists	417
BATCH Statement	418
BATCHPGM Statement	419
BROWSE Statement	420
CICSBMS Statement	421
CICSPGM Statement	421
CJOURNAL Statement	423
CQUEUE Statement	423
CREATE Statement	424
DATA ACCESS I/O Statement	424
DATABAS Statement	431
DATASET Statement	431
DB2 Statement	432
DCLCOL Statement	433
DELETE Statement	434
DEVICE Statement	434
DLIDSC Statement	435
DLIPSB Statement	437
DRIVER Statement	437
ERASE Statement	438
FIELD Statement	439
GETDIAG Statement	442
GROUP Statement	443
HOLD Statement	444
IMSDRV Statement	444
IMSMFS Statement	446

IMSPGM Statement	446
IMSPSB Statement	448
INREAD Statement	448
JOINCOL Statement	449
JOURNAL Statement	449
MATCH Statement	450
MATCHKEY Statement	451
MATCHX Statement	451
MERGE Statement	451
MERGE# Statement	451
MERGEGRP Statement	452
MERGEKEY Statement	452
NONTERM Statement	453
OUTREAD Statement	454
PANEL Statement	454
PARAM Statement	455
PLIXOPT Statement	456
READ Statement	456
READNEXT Statement	456
RECORD Statement	457
REPLACE Statement	458
REPORT Statement	458
ROW Statement	459
SCREEN Statement	460
SEGEDIT Statement	464
SEGEND Statement	465
SEGLOOP Statement	466
SEGMENT Statement	468
SORT Statement	469
SORTKEY Statement	470
SPBROWSE Statement	471
SPPARM Statement	471
SPPGM Statement	472
SPRDNEXT Statement	473
SPTRNACT Statement	473
SRC Statement	473
STORED Statement	474
STPROC Statement	475
TABLE Statement	476
TELON Statement	477
TLNJOIN Statement	477
TLNROW Statement	478

TPPCB Statement	478
TRANSACTION Statement	479
TSOPGM Statement	479
UPDATE Statement	479
WORKSPA Statement	480
XFEDIT Statement	480
Procedural Custom Code	482
CA Telon Source for a Screen Definition—COBOL	483
CA Telon Source for a Screen Definition—PL/I	485
Generate Processing Flow	486
Extraction of Exported Objects	486
Preprocessing of SCRNDEF with ADPCCARD	487
Generation of Shell Program	487
Extraction of Generated Objects	487
Resolving of Custom Code	488

Chapter 14: Generated Working Storage Variables 489

Alphabetical List of Field and Area Names	489
Section and Procedure Names	517
Online Programs	517
Batch Programs	520
CICS Nonterminal Programs	522
Stored Procedure Programs	523
PF Key Variables	524
DL/I Area Layouts	524
DL/I SSA Layouts	528
DL/I RSA Layouts	528
DL/I Linkage	531

Chapter 15: Advanced TDF Concepts 533

Using Base Definitions	533
BASE Panel Definitions	534
High Level Base Definitions	534
Creating and Using BASE Definitions	535
SEGLOOPing into a Table with Paging	535
Alternative Output Mapping in a SEGLOOP	538
Output SEGLOOP Browsing	538
SEGLOOP Parameters	539
Consistency Edits in SEGLOOPS	540
Referencing Screen Attributes in PL/I	541
Individual Field Edit Error Messages	541

Light Pen Support	542
SCREEN/EOFKEY	543
Application Work Area hhWKAREA	543
Program Custom Code Entry Point OINIT1	545
Program Custom Code Entry Point ININIT1	545
Generated Attribute Setting	546
Defining Fields	546
Sample Light Pen Program	548
Minimizing the Size of SPA/COMMAREAs	549
Decreasing the SPA/COMMAREA	549
Line Optimization and Screen Merge Processing	550
LINEOPT=1	550
LINEOPT=2	553
LINEOPT=3	553
CA Telon Screen Handling	553
Parameter Descriptions	553
Implementation	555
Using the Configuration Section	557
3270 Numeric Lock Feature	558
CA Telon Help Facility	558
CA Telon Screen-Level Help	559
CA Telon Field-Level Help	559

Appendix A: Using the IMS Transaction Manager Environment 561

Exiting IMS Application Programs	561
WORKSPA Database	562
Creating the WORKSPA Database	562
Defining a DB2 Table as a WORKSPA	564
Using the WORKSPA Database	564
Combined SPA/WORKSPA Database Handling	565
C-950-PUT-WORKSPA section	566
Dynamic Link Programs	566
IMS SPA/WORKSPA Database	566
JCL	569
Other Considerations	570
Static Link Program	570
Beginning and Ending a Conversation	571
HELP/HOLD Facilities	577
HELP/HOLD Databases	578
Setting Up TDF Help Panels	579
Setting Up the Help Panels	579
Summary	580

CA Telon-Supplied HELP Program	581
Writing Non-Screen Transactions	581
IMS/DC Report Processing	582
Report Definition	582
Report Program Structure	584
Controlling the Report Destination	585
Calling Program Interface	586
CA Telon Driver Applications	587
CA Telon Program Transfers	587
Message - Switch Transfer	588
CA Telon Dynamic-Link Transfer	588
CA Telon Static-Link Transfer	589
CA Telon Static/Dynamic-Link Transfer	590
Program Switching to Non-CA Telon Programs Using MFS	591
Using Multilingual MFS Screens	592
HELP Messages	592
Screen Literals	592
Error Messages	593

Appendix B: Using the CICS Online Environment **595**

Handling the Clear Key	595
Invoking a Nonterminal Program	596
Using Temporary Storage Instead of DFHCOMMAREA	596
CA Telon Program to/from Non-CA Telon Program	597
Accessing a Dynamically Loaded Table	599
CICS Driver Program	599

Appendix C: Using The Batch Environment **601**

Processing Passed Parameters	601
Sample Code	601
Batch Design Considerations for Packaging Reports	602
Example Batch Program	603
VSAM Example	603
IMS Example	604

Appendix D: Using The DB2 Environment **607**

Catalog Access Interface	608
Information Used by CA Telon	608
DB2 Catalog Structure	609
Using SEGEDITs Against SQL Tables	612
Linking Considerations	613

BROWSE Request	617
Plan Name Qualification	617
Fully Qualified Plan Names	617
Unqualified Plan Names	618
DB2 Synonyms	618
Application Coding Solutions	619
Executing CA Telon-Developed Applications	624
Pre-compile DB2 Applications	624
Bind DB2 Applications	625
Granting Public Access on Plans	627
FETCH Details—Using an Alternate Cursor	630
FETCH Details—FETCH for NN Rows	633
User DATATYPES	634
Declaring Global Temporary Tables	634

Appendix E: Using The DL/I Environment 637

BOOLEAN SSA and Secondary Indexes	637
Using Multiple PCBs for One Database	644

Appendix F: Importing Data Inheritance 645

Overall Approach and Recommendations	645
Proper Application Storage	646
Typical Import Usage	646
Factors Influencing Data Definition Stability	646
Input Parameters	647
Messages	648
Severity Codes	648
Reports	648
Recommendations for Import Processing	649
Production TDF Available	649
Empty TDF Available	650
Application Development	650
Importing an Object	650
CA Telon Definition Importing	650
Processing	651
Importing from a PDS	655
Importing from CA Panvalet	657
Importing from AllFusion Endeavor Change Manager	657
DL/I DBD Importing	659
Processing	660
Importing from a PDS	664

DL/I PSB Definition Importing	665
Processing	666
Job and Procedure—Importing from a PDS	667
Sample Report	667
Import Report Fields	669
Severity Codes	670
Messages	671
CA Telon Definition Import Messages	671
DL/I DBD Import Messages	675
 Appendix G: Using Stored Procedures	 677
Generated Programs	677
Stored Procedure Programs	677
Programs Which Call Stored Procedures	682
Calling "Foreign" Stored Procedures	687
 Glossary	 689
 Index	 703

Chapter 1: Introduction to Programming Concepts

This manual takes the application programmer step-by-step through the CA Telon Application Generator (CA Telon) program creation process. Detailed examples accompany each topic as you see how to create the panel image, panel definition, and program definition.

To use this manual you should have a basic knowledge of your operating environment and have COBOL or PL/I programming skills. Refer to the README for more on information resources.

Introduction

CA provides products that address the key areas of the system life cycle. These products perform analysis, design, construction, testing, implementation, and maintenance.

CA's approach provides an evolutionary code generation environment solution that significantly increases productivity in a controlled environment. This allows an organization to achieve the following objectives:

- **Productivity**—Increases the productivity to contribute to timely delivery of application systems. CA tools support rapid development techniques, including Joint Application Design (JAD), prototyping, and code generation.
- **Quality**—Delivers the right business solution that performs according to specification. CA's approach provides verification throughout the life cycle by enforcing:
 - Design rules, consistency/completeness checks, and agreement with standards
 - Technical requirements by allowing prototyping of models created during systems design
- **Control**—Controls the system development process from planning through the implementation phase, including the turnover of production systems.

CA Telon

One of CA's products that meets your system life cycle needs is CA Telon.

A Productivity Tool

CA Telon is a productivity tool that simplifies and speeds up the application development process, thereby increasing programmer productivity. Typical application development projects include online order entry systems, batch payroll systems, online inventory systems, and so on.

CA Telon automates many of the tedious and repetitive application development tasks, freeing you to concentrate on the application's external design and testing. CA Telon provides you with a highly efficient, flexible, and integrative tool for developing application systems.

CA Telon provides an integrated solution to the total application development process. Whether you create online or batch programs, CA Telon is powerful enough to support design and implementation standards presently used in your company. In addition, CA Telon's capabilities provide re-usable designs to meet your future needs as they develop.

A Tool that Meets Your Quality Requirements

CA Telon allows you to prototype program models to the application user before, during, and after code generation.

The code generation process itself produces well-structured, documented COBOL, or COBOL II and above or PL/I code.

A CASE Tool

CA Telon is a Computer-Assisted Software Engineering (CASE) tool that produces a completed application from beginning to end.

A Control Tool

As a control tool, CA Telon allows implementation of your MIS and development standards, such as program structure, naming conventions, and so on. CA Telon also allows sharing of databases between application programs and provides control of copy members.

Features and Benefits

CA Telon provides all the facilities necessary to bring an application from the early stages of design—through development and testing—to full production and maintenance. CA Telon delivers these productivity gains through the following functional areas:

Analysis Tool Interface

The open architecture of CA Telon enables other systems to interface their results into CA Telon to generate quality applications.

Screen/Report Painting

Painting screens and reports in free format simplifies BMS or MFS mapping statements and the underlying logic. With CA Telon, you can paint screens that comply with SAA CUA standards or your own installation standards. You can also copy the screens from master base screens to more easily establish consistency within an application or across applications.

Prototyping

CA Telon features a comprehensive modeling/prototyping facility. This allows you to create a working model from design-level information; a model that evolves into the production system. CA Telon's prototyping allows you to *animate* an application—complete with data, so that your application users can preview the screens before you have written one line of code. You can change screens during the prototyping session as needs or ideas dictate. This helps you to eliminate errors in the analysis and design phases of system development.

Correcting errors at this phase is less expensive than performing ongoing maintenance to an already constructed system. The Prototyping Facility is fully interactive and runs under TSO and CICS, and in PWS.

Data Dictionary Interface

CA Telon provides access to information stored in IBM's Data Dictionary, in MSP's Data Manager, and in PDSs by means of an import procedure. You can incorporate existing file definitions resident on the data dictionary, or in PDSs, into the generated application programs. CA Telon interfaces directly with the DB2 catalog to allow you to import table definitions into CA Telon.

File Generation/Groupings

You can import file definitions into CA Telon. You can alternatively define files in CA Telon. CA Telon provides you with the additional capability of grouping files logically into *file groups*. You can then add files and tables to programs on an ad hoc basis, or as logical units by copying these file groups.

Generated DBMS Access

CA Telon can generate I/O access to SEQUENTIAL, VSAM, GSAM, DL/I, and SQL. CA Telon uses generic terms to define access to each of the DBMS types, which facilitates easy conversion between different DBMSs. This also provides a buffering effect while programmers learn the new DBMS.

Design Capture

CA Telon captures external application design. It focuses on the external design characteristics of a program without introducing a new programming language. You enter design information on user-friendly, ISPF-like screens.

Flexibility

Many other high-level development systems require you to follow their development methodology. But CA Telon's development approach is adaptive. If you have an effective system development methodology already in place, you can choose which CA Telon techniques you will use and fit these to your existing techniques.

CA Telon's open architecture ensures flexibility. You can enter COBOL, COBOL II and above, or PL/I code into any program as custom code to enhance the business function. You can also reduce duplication of code by accessing common code stored in PDSs, or data dictionaries. Additionally, you can tailor CA Telon at installation to implement company standards by automatically including code in predetermined locations of a generated program.

Environment Conversion

CA Telon insulates you from environmental considerations (IMS and CICS) while creating the application. You spend more time developing the business functions correctly. You introduce environmental parameters at the end of the development cycle, at which point CA Telon generates all data communication access (screen and transactional I/O). You can easily convert from one online environment to another without changing the design information.

CA Telon Components and Program Development

CA Telon provides two interrelated components that address various phases of the application program development. These components are:

- The CA Telon Design Facility (TDF)—Allows you to enter high-level design specifications and maintain existing CA Telon programs
- The CA Telon Application Generator—Translates the TDF design information into COBOL or PL/I program source statements that run in your environment

CA Telon Design Facility (TDF)

The TDF is an ISPF-like interactive tool that you use to develop individual programs and complete application systems. The TDF runs in the following environments: IBM TSO, CICS, and Windows on an IBM Personal Computer (or compatible).

When creating programs using the TDF, follow these three basic steps:

1. Create a panel image (or an optional report image for batch)
2. Define the data to be used on the panel image or report
3. Specify the file and customized functions for the program type (screen, nonterminal, IMS/DC report or driver, or batch)

Panel Image (PI)

A panel image (PI) identifies the format of your screen or report. You create a panel image in one of three ways:

- Typing - *painting* - the screen or report image. You simply paint the literals (fields that supply information to the application user during program execution) directly onto a blank screen and use special characters to represent the data fields, the type, and the length.
- Copying a base panel or report from one already existing and modifying it as needed.
- Transporting in a screen or report definition created in another tool and modifying or enhancing it as needed.

At this point, CA Telon captures field locations, usages, and lengths.

```
>>>>>>> TELONSAMPLESOLUTION
```

	EMPLOYEE ID ++++++
1. NAME	<<<<<<<<<<<<<<<<<<<<<<
2. STREET	<<<<<<<<<<<<<<<<<<<<<<
3. CITY	<<<<<<<<<<<<<<<<<<<<<<
4. STATE	<<
5. ZIP	<<<<
6. PHONE	<<<<<<<<<<
7. SEX	<
8. DATE OF BIRTH	<<<<<<<
9. DEPARTMENT <<<	
10. DATE OF EMPLOYMENT <<<<<<<<<<<	
11. HOURLY RATE <<<<<<	
12. HOURS PER WEEK <<<<	

```
>>>>>>>>>>>>>>>  
-----  
  
PFKEYS: 2-HOLD 3-END 4-ENDHOLD 5-CANCEL 6-MENU
```

In the CA Telon Panel Image Editor, the > characters identify the output fields, < the input fields, and + a combination of output and input fields. You can define these characters both at installation time and at panel image creation time.

After you paint or edit a series of panel images, you can immediately enter a prototyping session to verify the screen designs and flow. Usually the next step, however, is to define the data edits associated with the screen. This step is called *creating the panel definition*.

Panel Definition

A panel definition (PD) is a collection of field edit and mapping destinations that define the appearance of a screen or report and the processing associated with it. It includes the panel image and the above field definitions.

Like the panel image, you can create the panel definition in one of three ways:

- Defining the characteristics of the screen or report
- Copying a base panel or report from one already existing and modifying it as necessary
- Transporting in a screen or report definition created in another tool and modifying or enhancing it as needed


```
TRCTDA.PD UPDATE PANEL,FIELDS *****
```

```
COMMAND ==> PAGE 01 MORE
```

```
OPTIONS ==> ATTRS   HELPMMSG _MAPOUT _
```

```
LINE 0007 COL 002 SIZE 024 000
```

```
---+-----1---+-----2---+-----3---+-----4---+-----5---+-----6---+-----7---+
```

```
0007 1. NAME <<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
0008 2. STREET <<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
0009 3. CITY <<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
-----
```

```
U LN COL LTH USE **NAME**FLDTYPE* ***** DBNAME OR TEXT *REQ MORE
```

```
01 003 000 0U DATE DATE XFER-TODAYS-DATE
```

```
04 040 006 0I ID XFER-EMPL-ID + +
```

```
07 027 025 IN NAME EMPL-NAME Y +
```

```
08 027 025 IN STREET EMPL-STREET +
```

```
09 027 025 IN CITY EMPL-CITY +
```

```
10 027 002 IN STATE ALPHA EMPL-STATE +
```

```
11 027 005 IN ZIP EMPL-ZIP +
```

```
12 027 012 IN PHONE EMPL-PHONE +
```

```
13 027 001 IN SEX EMPL-SEX +
```

```
14 027 008 IN DOB DATE EMPL-DOB Y +
```

```
15 027 003 IN DEPT EMPL-DEPARTMENT +
```

```
16 027 008 IN DOE DATE EMPL-DOE Y +
```

```
17 027 006 IN RATE NUMERICEMPL-HOURLY-RATE +
```

```
18 027 004 IN HOURS FLOAT EMPL-HOURS +
```

During this step, include any COBOL, COBOL II and above, or PL/I custom code to handle business requirements unique to your installation and to the program you are developing. You can easily enter this code, reference it in other programs, and maintain it at a later date using an ISPF-like editor.

For online programs, the program definition is called a *screen definition (SD)*. A sample Screen Definition Editor screen is shown.

TRCTDA.SD UPDATE SCREEN DEFINITION ** *****	
COMMAND ==> _____	
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV IMS _ SCRIN PARMS _	
STORED PROCEDURES	
GENERAL:	DESC MVS CICS TARGET - COB VSAM ADD _____ REMARKS **DFLT**
*	NEXTPGM _____ CURSOR NAME _____ SIZE 24 X 80 LANG COB (COB/PLI)
*	CMPLOPT _____ IDENTIF _____ PROCEDR _____
DATA	XFERWKA TRXFERW _____
AREAS: _	WKAREA **DFLT** _____
OUTPUT:	
A-100 _	OINIT1 _____ OINIT2 **DFLT** _ CURSCUS _____
B-100 _	OUTTERM _____
INPUT:	
P-100	PFKEYS ADD,ENT,1,2,3,4,5,6,OT _____
D-100 _	ININIT1 _____ ININIT2 **DFLT** _____
E-100 _	FLDEDIT _____
X-100 +	SCREEN XFEDIT/SEGEDIT _ CONSIS _____
H-100 _	INTERM **DFLT** _____
MISC: _	SECTION _____
*	PGMCUST _____

For batch programs, the program definition is called a batch definition. This is created in a process similar to that of online programs. You paint report layouts using the same panel image and panel definition procedures used in online development. You then define a batch definition (BD) instead of an SD.

When you complete the screen or batch definition, the captured design level information is ready to be generated into a native source program by the CA Telon Application Generator.

CA Telon Application Generator

The CA Telon Generator is executed using a batch job to produce COBOL, COBOL II and above, or PL/I source code for application testing or production execution.

Screen Definition Input

Before the actual generation of native code occurs, *export* of the program specification occurs to update an intermediate file called *CA Telon code*. The export step takes all of your definitions (PI, PD, SD, RD, DR, ND, BD, or SP) and creates input to the CA Telon generator.

Generator Control

The generator is statement-driven. The CA Telon code statements are fed into the generator, which produces the native code. CA Telon specifies the structure (for example, screen, nonterminal, report, driver, or batch) and target environment (for example, CICS or IMS) for the generated program based on the type of program definition created and by the environment statement associated with that program definition.

During testing, you can generate an application as a self-contained unit without BMS control blocks for ease of development, and later generate it with interfaces to BMS along with the BMS source code. The CA Telon administrator or technical support person can accomplish these changes without any involvement by the application development group.

Program Structure

A CA Telon-generated program is easy to follow, partly because it conforms to a standard hierarchical structure. In addition, your custom COBOL, COBOL II and above, or PL/I statements are inserted into the generated program just as you coded them. This standard structure provides high development and maintenance productivity without sacrificing performance.

The generated program also contains calls to CA Telon-delivered subroutines used for editing and optimizing line traffic. Some of the processes performed by these routines are:

- Syntax editing and formatting of screen or report variables
- Optimization of line traffic for online screen programs
- Specialized ABEND-handling routines for controlled application ABENDS
- CICS screen-mapping routines

CA provides source code for all editing and ABEND-handling routines. You can also generate source code in the CA Telon application to handle the screen line optimization processing. This optimization processing allows you to continue maintenance of the application even if you discontinue using CA Telon. You can also customize applications for your own environment, by creating additional syntax editing and ABEND-handling routines that will be called by the CA Telon-generated code.

Since CA Telon generates a native COBOL, COBOL II, or PL/I source program, no CA Telon run-time monitor or other *black box* is necessary for program compilation or execution.

Online Program Architecture

In an online application, you accomplish tasks through a series of related screens. You must look at the CA Telon screens you design with this output/input architecture in mind. Each online CA Telon program has two distinct parts:

- **Output**—Where fields are reformatted and mapped from work areas, DB2 tables, VSAM records, or DL/I segments to the screen
- **Input**—Where fields are edited and mapped from the screen to work areas, DB2 tables, VSAM records, or DL/I segments

CA Telon online programs, therefore, are structured to format the output screen, write to the terminal, read from the terminal, then process the input from the screen.

This output/input architecture is the foundation of all other CA Telon concepts. All CA Telon statements define action specific to screen output or input processing.

This output/input architecture philosophy differs slightly from the standard CICS design philosophy where the physical architecture of a program usually centers around the transaction, which generally inputs from one screen and outputs to another. It is also somewhat different from IMS/DC where you generate a separate program for each activity.

Part of CA Telon training is the understanding of the basic CA Telon design unit, the physical program structure, and the thought process used to develop and design a CA Telon application.

Batch Program Architecture

CA Telon Batch addresses both applications with report output and applications with significant database or file access.

In each of these, CA Telon allows you to select which of four program substructures that will best fit your needs:

- Simple sequential transaction input
- Transaction input requiring a mainline sort
- Multiple transaction inputs requiring merging
- Transaction and master inputs requiring matching

Program Design Steps

Like online program design, there are three steps in the design of a batch program:

1. Design the report (output-PI)
2. Define the data associated with the report (PD)
3. Complete the program definition by defining the file(s) to be used and any customized processing to be used

If your report is secondary to the processing, you can change the above order to perform step 3 first. If your application does not include report output, you perform only step 3.

Note: While the physical program for batch is different than the online program, the steps used to create a CA Telon batch program are similar to those used to create an online CA Telon program. In this way, your CA Telon skills apply to both Batch and online programming.

Part of CA Telon training is to teach how to understand and use the powerful CA Telon capabilities in both batch and online.

CA Telon Specifications

CA Telon provides all the facilities needed to bring an application from the early stages of design through development and testing to full production and maintenance. This subject provides an overview of these facilities.

Environments

CA Telon operates in a variety of environments on the mainframe and PC.

Development Environments

The CA Telon Design Facility (TDF) is where applications are developed. CA Telon's Prototyping Facility is part of the TDF Environment. The TDF operates under the following environments:

- TSO
- CICS
- Windows

Generator Environments

The CA Telon Application Generator is where COBOL, COBOL II, and PL/I code is automatically generated. The CA Telon Application Generator operates in the following environments:

- TSO
- Windows

Target Environments

The CA Telon Application Generator automatically generates code for the target environment specified. The CA Telon Application Generator can target programs for the following environments:

- IMS/DC
- CICS
- batch

Databases

CA Telon generates data accesses to any combination of the following database management systems:

- &U\$TNDATCM.
- &U\$TNIDMSD.
- DB2
- IMS/DB (DL/I)
- VSAM
- Sequential
- CICS queues
- CICS journals

Languages

The CA Telon Application Generator translates the TDF design information into the following source programming languages:

- COBOL II
- COBOL for OS/390 and above
- PL/I

Interfaces

CA Telon has a Transport Facility that enables any vendor or user to create an interface between CA Telon and other products: CASE Front-End tools, change management packages, configuration management packages, and so on.

Data Dictionary Interface

CA Telon uses an import procedure to provide access to information stored in IBM's Data Dictionary, in MSP's Data Manager, and in PDSs. File definitions that already reside on data dictionaries or PDSs can then be incorporated into the generated application programs. CA Telon interfaces directly with the DB2 catalog to allow table definitions to be imported to CA Telon.

Chapter 2: Designing CA Telon Programs

CA Telon allows you the freedom to design your CA Telon programs using any methodology available for your shop. Generally, however, you should use the techniques outlined in this chapter for creating your CA Telon generated programs. These techniques tie together the CA Telon Design Facility (TDF) and the generator into an integrated unit.

Note: Although these design techniques are recommended for CA Telon, you can use any standard system development methodology. You can also choose which, if any, techniques to include in your methodology.

This chapter discusses, in detail, designing programs with CA Telon online and batch.

Designing with CA Telon Online

The concepts of screens, screen flow diagrams, design-time prototyping, and evolutionary development are important to consider when using CA Telon to develop online programs.

Screen concepts

The screen is a basic design unit in CA Telon. The examples used in this guide center around several screens. You design each one as a separate unit. You work with an add screen, a display screen, and so on. For information on screen types, see "Screen Concepts" later in this chapter.

Screen flow diagram

With the screen flow diagram, you tie the details of several screens into one system. When you have many screens, you can break your flow diagram into groups of flow diagrams.

When designing programs, screens such as the menu, add, update, display, and list have similar functions and should be grouped in a single flow diagram. On the other hand, various screens concerned with printing reports or analyzing time usage should be in separate flow diagrams.

Design-time modeling

With design-time modeling, you create models of your application. Using the model with the CA Telon Prototyping Facility, you can realistically simulate the final version of your system down to accessing real files. This model allows you to demonstrate the system to the application user during the design stage when you can more easily make necessary changes.

CA Telon's unique modeling and prototyping function allows you to create a model which then automatically becomes the generated program. With CA Telon, prototyping is fast, easy, and not throw-away.

Evolutionary development

Evolutionary development is merely an extension of design-time modeling.

A particular system can have a series of two or more models depending on its complexity. The final model becomes the implemented version.

Extending this one step further, the present implemented version becomes the model for further development when it comes time for maintenance to update the system. You build on what you already have. You do not have to start over from scratch.

Screen Concepts

The screen concept is an essential concept for the use of CA Telon. It differs slightly from the traditional transaction concept of IMS or CICS where attention centers around the transaction, which is usually the input from one screen and the output to multiple other screens. In the CA Telon screen concept, attention centers around the screen, output to the screen and input from the screen. The performance of a user function defines which screens are presented to the application user and their presentation sequence. Therefore, user function considerations include:

- On what screen is the function initiated?
- What additional screens (if any) are executed in the implementation?
- What screen is presented to the application user on function conclusion?

Note: Multiple functions are often initiated from the same screen while other functions are concluded with the same screen.

Screen Types

When using the screen concept, you often want to design screens to perform five types of functions. Often, a screen includes one of these functions. At other times, a screen includes multiple functions. Generally, the user interface is more simple and friendly, when multiple functions are not combined on the same screen. A definition of these screen functions follows:

- **Search screen**—A screen (usually a menu screen) on which a data key is entered to access a database or file. With CA Telon, you can easily add a consistency edit in the screen process to ensure that the key entered is consistent (for example, valid) with the information in the database. If the key is consistent, control usually transfers to a display, update, add, or list screen. If it is not consistent, CA Telon automatically returns an error message to the application user and is ready for a correction and re-enter.
- **Add screen**—Used to add (create) new records to the database or file. The screen may initially display blank fields. Often, an Add screen includes update functions as it adds certain database records and updates others.
- **Display screen**—Simply displays data, database or file records in usually protected fields. A display screen is typically a place for the application user to make processing decisions. Therefore, display screens often include select options, used by the application user to select action to be taken on a different screen or in a different function.
- **Update screen**—Used to update existing records in the file. It displays unprotected data, allowing the application user to key over the data.
- **List screen**—Provides for the add/display/update of multiple occurrences of the same data. It usually includes the ability to page (scroll) forward and backward as well as the ability to select a particular entry for further action (such as display or update).

You can easily include several functions (such as search, add, display and update) on the same screen, known as a multi-function screen. However, single-function screens may be easier to create, modify, and understand than are multi-function screens.

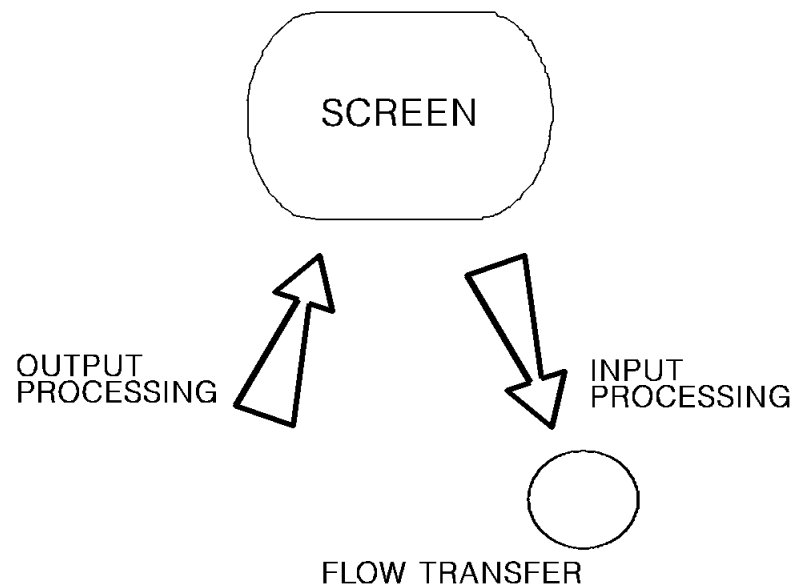
Note that menu and delete screens are not considered as one of the five functions. This is because a menu screen (if it is to be effective) is really a search screen, providing the search against multiple types of data. The delete function is usually just an option from a list, display, or update screen.

Screen Concept Components

Analysis and design of a screen under this concept involves thinking of the screen as having three basic components, described below:

1. **Output**—The screen is presented to the application user. This operation could be trivial (for example, when an empty format is presented to the application user). More often, however, it involves reading database records, computing values, and editing fields as they are mapped from source fields to the screen for display to the application user.
2. **Input**—The data is read from the terminal and application-user actions are determined. Input fields from the terminal are edited, converted, and mapped into destination fields in database records or work areas. If errors occur, CA Telon automatically highlights the fields in error and returns an error message to the screen. If no errors occur, data is processed and database records can be created or updated.
3. **Program or flow transfer**—After input processing is complete, a decision is made as to which screen receives control to continue the logical function. Sometimes this is trivial, as when screen one always transfers to screen two. Usually, however, some action from the application user (for example, a value entered, a PF key pressed) determines which screen gains control.

Screen components



Screen Fields

Using the screen concept, you can best analyze and document a screen by specifying each variable field to be used as an output, input, output-input (or outin), or select field (defined below). This assists the analysis of the functions of the screen and documents its intent. Even when a single screen is used as a combination add/change screen (either function on the same screen), it is often desirable to lay out separate screens to document the different uses. (It does not necessarily follow that all outin fields on a change screen become input fields on an add screen.) Definitions of the five field types are as follows:

- **LITERAL**—Any character or series of characters that are not used in processing but only supply information to the application user. Headings are a common type of literal.
- **INPUT**—Any character or series of characters that are to be keyed by the application user for the first time and processed as input from the screen. In most cases, the field is mapped to a database, file, or work area field, but the field can also be used in various types of procedural logic. An input field is defined on a screen layout by the less-than character (<).

An input field usually starts as spaces, but options exist in CA Telon to allow an input field to be initialized to nulls (for extra line optimization) or to underscores (so that the application user sees the size of the field). Additionally, you can initialize an input field to any constant.

- **OUTPUT**—Any character or series of characters that is used in processing and is displayed to the application user but cannot be modified. An output field differs from a literal field in that its value is determined either from a field on a database, file, or work area or from special processing by the program. An output field is defined on a screen layout by the greater-than character (>).
- **OUTIN**—A combination of an output and an input field. It is processed on output by the program and is modifiable; that is, it can be keyed into by the application user and is processed on input by the program. An outin field is defined on a screen layout by the plus sign (+).
- **SELECT**—A special type of input field that is used by the program to allow the application user to choose a processing path from several options. Its use can simplify later programming. When used for a screen, the application user must enter data in one and only one of the SELECT fields to determine the system action to be taken. Usually a SELECT option is a single-byte field that allows any non-blank character for selection. Other times, however, it can be multiple characters and require a certain format or set of values (for example, line number on a list screen). A SELECT field is defined on a screen layout by the vertical bar (|).

These field types are discussed further in the chapter "Creating a Panel Image."

Screen Flow and Documentation

You can use the screen flow and documentation technique to evaluate and document external system design. The flow diagram is first used for conceptualizing a design and initiating questions and alternatives. It is used, along with the screen concept, to further expand the design, add additional detail, and evaluate system design against requested user functions. In addition to conceptualizing design, this technique documents the design for programming and further evaluation. Additional considerations for conceptualizing and documenting a design are presented below.

Circle Concept

The circle concept is a logical extension to the screen concept. When a project is large (more than 15-20 screens), it can be difficult to design and evaluate a single screen flow. Often, it helps if you break up the screens into groups of screens called *circles*, which are related to each other by function. (The most important characteristic of a circle is that its screens pass control mainly to other screens in the same circle.) You can then view the application first in terms of circles, and then, in each circle, in terms of individual related screens.

Screen Flow—Circle Diagram

The screen flow can assist you in conceptualizing the online or batch design. Important characteristics of this approach are as follows:

- Key issues in the screen flow are whether screens are connected correctly to perform user functions, mirror the application user work flow, simplify and optimize performance of the most common functions, and avoid the execution of invalid paths.
- The screen flow diagram can be used to check the execution of each function by analyzing the path it takes. For each function, you trace and evaluate the screen path necessary to perform that function for effective application user execution. You should also evaluate the transition from function to function by tracing paths.

- The objective in screen flow design is not to minimize the number of screens, which is sometimes done in traditional development where considerable effort is required for each extra screen design. Instead, the object is to customize screens to simplify user understanding and to maximize application user performance and productivity.

In general, minimizing the number of screen iterations an application user must make to accomplish the required work produces the optimal design, from both a user and a machine point of view.

- The flow of an application system does not have to be the same for all users. Instead, you can customize some flow characteristics depending on the application user or the present mode of operation of that application user. For example, if the application user is in add mode, the add screen might transfer to itself, whereas if the application user is in random function mode, the add screen might transfer back to a menu screen.

Evaluating a Design

The design process entails multiple iterations of conceptualizing and evaluating designs. The screen flow, initially used for conceptualizing a design, should also be used in the evaluation of that design. Use the screen flow diagram and screen layouts together in the predesign-prototype phase (see "Design-Time Prototyping" later in this chapter) to analyze the design against the application requirements. You can do this by working various application function scenarios through the flow and screen designs. Give special attention to the criteria listed below. Following that evaluation, you can create a design-time prototype to test the effectiveness of the application against those scenarios in live, online processing mode.

- In scenario evaluation, the screen flow must accurately reflect the new work flow anticipated by the application user.
- The objective is to optimize the processing of normal functions, with less emphasis on exception handling. Although carrying the application user through extra screens is not satisfactory during normal processing, it is allowable for exceptions.
- The application system should be designed to handle different kinds of users (for example, experienced versus inexperienced) with equal effectiveness. For example, the use of the HELP Facility is available to assist a new application user, but does not encumber an experienced one.
- The number of screens used to process a function should be minimized, as long as it does not create complications in the users' understanding. A trade-off to be evaluated in design often is the simplicity gained by breaking apart functions into multiple screens versus the efficiency gained by combining them.

- If the application involves a new database design, that design should take into account application requirements as determined in the screen flow and screen layout.
- During screen design, give special attention to the readability and clarity of the layout. Again, you must trade off the efficiency of combining many fields on a screen versus the readability gained by breaking them apart. You should also give attention to cursor movement. The cursor should move left-to-right and then top-to-bottom. If INPUT fields are aligned in the same column, the application user can more easily follow the movement of the cursor as it goes from field to field.

Documenting the Design

You can document the application design through a combination of the screen flow diagram, screen layouts, screen map/edit charts, and screen narratives, described individually below.

Screen flow diagram

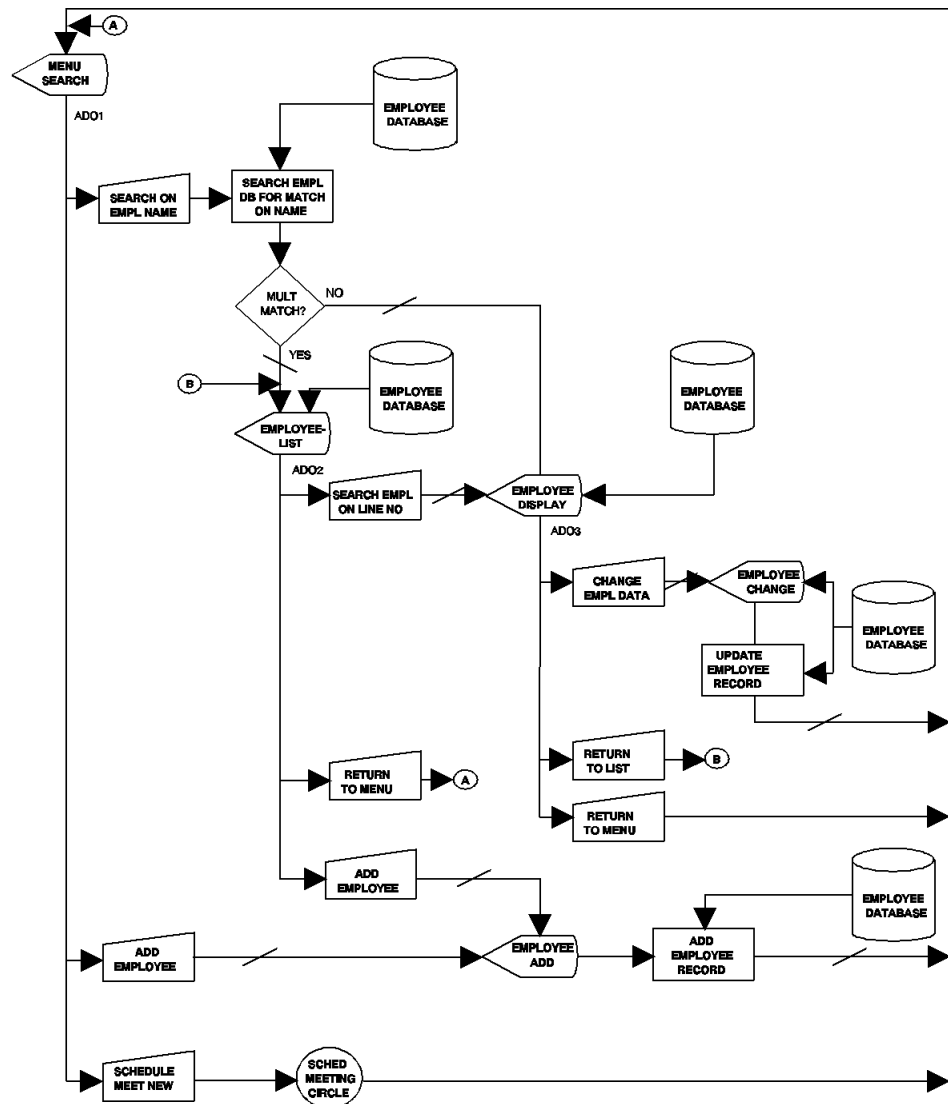
The screen flow diagram documents the various screens, the database/files they access, and the flow from screen to screen. If the flow diagram is clear and accurate, you can trace the sequence of screens required to perform functions and determine if that flow reflects and optimizes the performance of user functions.

The screen flow diagram shows a single circle that performs general maintenance. When processing in the maintenance circle completes, control passes to the SCHEDULE MEETING circle (named at the bottom of the diagram).

The conventions used in conjunction with the diagram are detailed in the Diagram Symbols chart. You should follow these conventions when charting groups of application screens in a screen flow diagram. Other general rules include:

- Show only high-level processing with only minimal detail.
- Omit error iterations (these are assumed for all screens).
- Emphasize the screen-to-screen (and circle-to-circle) flow.
- Show where data accesses occur, either on the output or input side.
- Describe variable flow using either the SELECT option symbol, if SELECT fields are used on the screen, or the decision block, if flow is determined by IF/THEN/ELSE logic in procedural code.
- Omit HELP Facility screens in the flow since any screen can transfer to HELP.

- If the application uses standard PF keys (for example, a PF key that causes the same action to be taken on every screen where it is active), do not document those PF keys that cause flow transfer on the flowchart. Instead, define them in a separate document. You can indicate the PF keys that are active on a screen by simply listing them below the screen symbol on the flowchart. On the other hand, if a PF key causes flow transfer that is unique to a screen, document it with a decision block, as you do for other variable transfers.



The following table describes each circle diagram symbol.

Diagram symbols

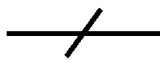
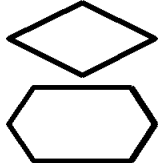
SYMBOL	DESCRIPTION
	The Screen Symbol represents an application screen. For CA-TELON, a horizontal line is added to the right edge of the symbol to denote the output/input architecture: all logic or I/O shown above the line represents output processing; logic and I/O shown below the line represents input processing. For the application user, this line represents the point where the Enter key is pressed. For the programmer/analyst, this line represents transaction scheduling.
	The Processing-Flow Symbol represents operations sequence and dataflow direction.
	The Transfer Symbol - a slash across a processing flow symbol - represents the separation of input processing for one screen's program from output processing for the next screen's program.
	The Entry/Exit Symbol represents entry into or exit from a circle. A circle can have multiple entry points but only one exit point.
	The Select Symbol represents a decision based on a CA-TELON SELECT field, which is used to control processing flow.

Diagram symbols

SYMBOL	DESCRIPTION
	The two Decision Symbols are used interchangeably and each represents a decision. Two lines emerge from each decision symbol: one marked YES (generally first moving clockwise) and one marked NO (second moving clockwise). These lines correspond, respectively, to the logic that follows if the condition being tested is true or false. It is preferable to use the top symbol; however, if you want to include more information than can fit in that symbol, use the bottom symbol.
	The Processing Symbol represents Custom Code processing on a conceptual (i.e., general design) level.
	The Database or File Symbol represents I/O associated with the screen. Arrows running to the symbol represent output; arrows running from the symbol represent input.
	The Circle Symbol represents entry into another circle, which is named within the symbol. Note that this name must match a name at the top of the appropriate circle diagram.
	The Onpage Connector Symbol represents entry into or exit from another part of the Circle Diagram on the same page.
	The Offpage Connector Symbol represents entry into or exit from another page of the Circle Diagram.

Screen Layout

The following example illustrates a screen layout. You create a screen layout for each screen defined in the flowchart. Sometimes each field is numbered to correlate it with the same field on the Screen Map/Edit chart.

[illegible]

Screen Map/Edit Chart

Under some circumstances, it is desirable to create a screen map/edit chart prior to the creation of a panel definition.

A screen map/edit chart describes the editing and mapping for each field on the screen. You define the edit to be performed for each field, along with an indication, for INPUT and OUTIN fields, of whether the field must be entered. The screen map/edit chart defines mapping by providing the source and/or destination field (in the program work area, transfer work area, or database/file layout) for each field on the screen. Define other unique characteristics of a field here also.

SCREEN MAP/EDIT CHART

HEADER: DC SCREEN ID: MR1A DESCRIPTION: MR SYSTEM - NEW MENU
USER DEFINER: BILL ROSS ANALYST: KEN CARLSON

#	Field Name	Field Edit	Use	Req	Description	Special Considerations
1	ROOMCNT	Numeric	O	N	S-ROOM-COUNT-	

#	Field Name	Field Edit	Use	Req	Description	Special Considerations
					RECORD	
					D-	
2	ADDOPT		S		S-	
					D-	
3	DISOPT		S		S-	
					D-	
4	UPDOPT		S		S-	
					D-	
5	DELOPT		S		S-	
					D-	
6	OPER	OPERFMT	I	Y	S-	
					D-OPER-ID	
7	OPERPW	None	I		S-	Non-display field used in consis edit
					D-	
8	ROOMID	Character	OI	Y	S-	
		numeric			D-XFER-WORK-FIELD	Passed to other screens
9	ERRMSG1	None	O		S-	
					D-	
					S-	
					D-	
					S-	
					D-	
					S-	
					D-	

A blank Screen Map/Edit chart, which you can copy and use follows.

SCREEN MAP/EDIT CHART

HEADER: _____ SCREEN ID: _____ DESCRIPTION: _____
 _____ USER DEFINER: _____ ANALYST: _____

#	Field Name	Field Edit	Use	Req	Description	Special Considerations
1					S-	
					D-	
2					S-	
					D-	
3					S-	
					D-	
4					S-	
					D-	
5					S-	
					D-	
6					S-	
					D-	
7					S-	
					D-	
8					S-	
					D-	
9					S-	
					D-	
10					S-	
					D-	
11					S-	
					D-	
12					S-	
					D-	

#	Field Name	Field Edit	Use	Req	Description	Special Considerations
13					S-	
					D-	
14					S-	
					D-	
15					S-	
					D-	
16					S-	
					D-	

Screen Narrative

The screen narrative defines any additional information not apparent from the screen flow diagram, the screen layout, or the screen map/edit chart. Break out the narrative into the outline described below and write it in structured English. Direct it to both the application user (as part of the written documentation for evaluation) and to the system implementer (for conversion into the CA Telon screen definition). Note that the outline is separated into the two parts, output and input, that always make up the analysis or implementation of a CA Telon program.

Output processing outline

- List of DB segments or logical entities or files accessed
- Special processing required in formatting output (for example, under certain conditions, a field on the screen is protected from modification)

Input processing outline

- List of each consistency edit (for example, cross field validation, database validation). If SELECT fields exist on the screen, define consistency edits under the select option for which they apply.
- Special processing to be done on input (for example, field initialization, calculations).
- List of DB segments or files created or updated. If special considerations affect that activity, describe them here.
- Description of flow transfer activity, including any special logic affecting that flow.

Transition to Program

The documentation described above represents external design (the design as the application user sees it externally), not internal structures used to implement that design. Under CA Telon, this external design is usually adequate for the transition from design to programming. That is, most of the tasks and documentation associated with traditional detail design (detail specifications) are not required. A strong relationship exists between the documentation described above and the screen definition. This simplifies the task of interpreting the design and converting it into the parameters required in the screen definition. For both the design documentation and the programming implementation, the process revolves around the concept of screen output and screen input.

Design-Time Prototyping

Prototyping is an external design and analysis technique that helps you:

- Communicate design ideas
- Evaluate the effectiveness of design decisions
- Prompt new design thinking
- Enhance the decision-making process

The application development process using prototyping is a combination of both function and data. Function relates to what the application user can do at a particular point in the work flow. Data is those elements related to the execution of the function. Thus the prototyping technique is neither process driven, where each function is broken into parts without regard to data, nor data driven, where the data is analyzed without regard to the functions to be performed.

In operation, prototyping techniques provide for the evolution of both function and data. Initial screen flows, screen functions, or screen data do not have to be complete. Rather, they represent design concepts at various stages in the design process and evolve to completeness as the design progresses. During this evolution, considerable design focus must be placed on work flow and how effectively the function and data represented by the screen design reflect that work flow.

The prototyping techniques do not replace the standard data analysis process, where data is gathered and organized by its inherent structure without regard to the application process. In fact, that data structure analysis feeds the application process while at the same time the application process feeds the data structure analysis.

Prototyping is not a rigid process involving the same prescribed steps for all situations. The type of prototype selected and where, during the application development process, that prototype is used differs depending on the requirements of the application and the preferences of the developer. For example, an early stage of prototyping can involve laying out sample screens with the application user, immediately reviewing their use with the Prototyping Facility, and executing application scenarios (for example, screen sequences) to test the proposed design against anticipated work flows. This gradually evolves into a final prototype that includes the complete working of all application functions against real databases. Prototyping is discussed in detail in the chapter "Prototyping Facility."

Evolutionary Development

This section discusses the advantages of using evolutionary development method.

Decision Making

In the traditional System Development Life Cycle (SDLC) approach, all decisions applicable to the level of one phase must be made then and are not to be changed in later phases. This makes decision making slow and uncertain since adequate information often is not available to make a decision, and the inability to change that decision requires longer scrutiny.

In evolutionary development, you always get two chances to make a decision. You make the initial decision during one phase, evaluate the effectiveness of that decision through the prototype, and then get an opportunity to modify that decision in the next phase.

Decisions that require additional information are postponed to a later phase when prototype usage and additional analysis provide that feedback. Care must be taken, however, so that decisions already passing two evaluations are not modified again and again during later phases.

Evaluation and Documentation

In the traditional SDLC approach, each phase represents a different phase of evaluation and documentation from the former (detailed design specifications versus general design specifications). In evolutionary development, the evaluation and documentation are the same for each phase (see *Circle Flow Concept*). What differs is the scope of each phase and the amount of detail addressed.

The first phase addresses only the heart or main application area and has little editing, calculations, and other special processing. A prototype of this design is then created.

The second phase lets you modify decisions made in the first phase (based on prototype feedback), add more edits and special processing, and address new areas of the application (but again without much detail). Again, a prototype is created and this concept continued until the final system is completed.

The number of phases (cycles) is variable and depends on the complexity of the application. Simple applications have few phases (generally two), while complex applications have several (three to five).

Maintenance

Using this technique, maintenance is just another cycle, in which the current production system is the prior prototype undergoing analysis and modification.

System Development Life Cycle Alterations

When you use CA Telon prototyping, as explained earlier in this chapter, the SDLC can be altered to take CA of the different prototyping levels. When doing this, you need to break the cycle into multiple phases, where each phase is defined with respect to the following criteria:

1. **Scope of business function**—The amount of business function that should be included in a phase. In CA Telon terms, this is identifying which circles should be implemented. Considerations guiding this decision follow:
 - The scope of business function should be limited enough to put reasonable time constraints on this phase (two to six months).
 - The phase should include enough meaningful business function that the working prototype is of value in itself and can be tested thoroughly.
 - The business function implemented should be mostly independent from other functions or circles. That is, changing designs for other functions should have minimal impact on this function.
2. **Level of design detail**—The amount of application detail that should be included in each phase. This should be considered with respect to these CA Telon design components:
 - Circle flowcharts
 - Panel images
 - Panel definitions
 - Local view of data structures
 - Program definitions
 - Algorithm definitions, CA Telon custom code

3. **Level of prototype**—The CA Telon prototyping level that should be used to most effectively communicate design information and verify the effectiveness of that design. Levels include:
- Prototyping without data mapping
 - Prototyping with data mapping, field editing, and flow control
 - Screen execution without data access
 - Screen execution with generic data access
 - Screen execution with data access

There is a strong relationship between the design level and the prototyping level. For example, prototyping without data mapping is used in conjunction with a panel image.

An application phase can be considered complete when a specific amount of business function has been defined using specific CA Telon design components verified with a prototype to the application user's satisfaction with a particular prototyping level. For example, the initial phase for a set of business functions could require the circle flowcharts, panel images, panel definitions, the local view of data structures, and related documentation prose (written explanations as opposed to design documentation captured automatically by CA Telon). The phase would be complete when those components have been created and sample application scenarios reviewed with the application user via prototyping with data mapping.

An alternate for this phase could include the same design components, but require an interactive prototype using screen execution with generic data access. The level of detail of application definition and the prototyping level used to verify its validity can vary from project to project, but must be clearly defined for any specific project.

The design components and prototyping levels mentioned above are described earlier in this chapter.

Designing with CA Telon Batch

Batch applications can vary significantly. Therefore, no single design approach is applicable for all batch applications. However, practical experience shows that typical batch processing includes the following common characteristics:

- Frequent data access
- Reporting
- Data sorting
- Data merging from multiple sources
- Matching transactions to a master

CA Telon batch handles all these batch processes.

CA recommends you use a modular approach to develop CA Telon batch systems. You should partition large systems into the smaller, isolated modules of definition, programming, and testing. Then, at a later time, you can integrate these components into one whole system.

This modular approach has the following benefits:

- **Simplified initial development**—Simplifies the initial development of a complex system.
- **Easier maintenance effort**—When an original system requirement changes, only the module addressing that requirement needs to be analyzed and modified for the new requirement.
- **Flexible component packaging**—The modular approach facilitates additional flexibility in the packaging of the components. This can be especially important for restart/recovery issues.

Systems with Reports

When developing a batch program with significant report output, you should focus on output design first since a batch report generally reflects specific end-user requirements. Therefore, you first create a panel image to *paint* the report. This is further defined with a panel definition. Then you create a batch definition where you address file/database access and special processing needed.

When the report is of secondary importance to the function (processing) of the program, you can alter the order of design and create the report layout last.

Although an order is specified in the definition steps below, the development process is iterative and generally requires switching among all of these steps. You can also start the process at any definition step that seems appropriate for the particular problem.

In specifying a batch program with a report, you execute the following four steps:

1. Report design (output)
2. Transaction design (input)
3. Database/data set definition
4. Additional processing logic (process)

Step 1: Report Design (Output)

This is the first step in developing a batch report. You specify the *image* of the report by using the Edit Panel Image screen in the TDF. Note that the layout of a report page is not simply painted as it exists when printed. This is because each page of a report is not necessarily formatted exactly the same as others. Rather, report pages are usually made up of *groups* of repeating information.

For example, the page heading lines represent a *group* of information that repeats at the top of every page. Another *group* is usually detail reporting information that repeats once for every *transaction* input driving the report. A third *group* could be totalling information that repeats every time some characteristic of the input changes (for example, a division or department change).

The panel image is the image of each of these groups of repeating report information. The batch program controls the order in which the report presents these repeating groups. After specifying the image of the report, you can then specify information about the fields (for example, mapping, formatting, group types).

Step 2: Transaction Design (Input)

The second step of a batch design is to specify the input or inputs that are to be the driving force of the batch program. The input driving force is called a transaction. In its simplest form, a transaction can be serial input from one or more files or databases (for example, sequential or VSAM files, DL/I, or DB2 databases). Other simple transaction input sources are IMS/DC message queues, interfaces to other programs, and JCL parms.

When processing simple transaction input, you can implement reusable transaction input by reconfiguring the system after testing. Do this by changing the simple transaction input to a program-subprogram interface. Create a driving program performing the transaction input, and remove the transaction input from the existing processing programs. Convert the processing programs to subroutines, and the transaction input is now reusable.

You can define more complex transactions for synchronizing multiple inputs. *Synchronizing* of the inputs depends on identifying the common input characteristics. For example, assume that all inputs contain an employee identifier. When reading the inputs, synchronization should occur for the employee name (part of identifier) of each input. The transaction in this case is a record area (or areas) that is to drive one iteration of the batch process and usually generate one or more lines of report output.

If input synchronization requires that you match a *transaction* input against a driver (or *master*) input, use the CA Telon match feature. If synchronization requires that you treat multiple inputs logically as one, with a record from one of the input files at a time being identified as *current*, use the CA Telon merge feature.

If input synchronization requires sorting before, during, or after processing, use the CA Telon mainline sort or user sort features. For a description of the data access features of CA Telon you can use for transaction input processing, see Chapter 6, "Creating the Panel Definition" and refer to the *Design Facility Reference*.

Note: You *must* design the transaction input early in the batch definition phase because the structure (or resultant program) is based on the CA Telon transaction feature selected. Simple transaction input, match, merge, and mainline sort processing, each result in a different generated program structure.

For more information on batch program structures, see Chapter 7, "Program Hierarchical Structure".

Step 3: Database/Data Set Definition

This third step of a batch design allows you to identify the database(s)/data set(s) to be accessed by the program. You identify their characteristics and the type of access (for example, read, write, update) using the TDF user exec definition screens (just as for online systems). CA Telon then performs the access from custom code.

Step 4: Additional Processing Logic (Process)

The fourth step is not required for simple programs that just read a sequential file and write a report. However, if any additional function is to be included or if special logic is necessary to control the report contents, you must add processing logic through custom code.

The program logical structure is simply:

- INIT—Initialization
- INPUT—The transaction access
- PROCESS—Custom code functional logic
- OUTPUT—Writing report lines
- TERM—Termination processing

Note that the processing logic can be added without regard for the report formatting considerations. Page and control break handling occur automatically during the output stages and are isolated from process.

Note: The report does not have to be directed to a print device. The report can be directed to an sequential file, VSAM file, or GSAM file.

Although there are multiple points at which processing logic can be added, most custom code is added at the PROCESS logical point. You can incorporate user sort processing at any logic point in the routine. To sort a file before processing the file, implement a user sort in INIT processing. To sort output data (including reports), implement a user sort in either PROCESS or TERM.

Systems without Printed Reports

In developing a batch program with no report output or insignificant amounts of report output, you do not create a panel definition initially. However, the logic of the next several steps is the same as that above when a report exists. Thus, you create a batch definition first, where you address file and database access and special processing.

Although the definition steps below specify an order, the development process is iterative and generally requires switching among all of the steps. You can also start the process at any definition step depending on what seems to make most sense for the particular problem.

In specifying a batch program without a report, you execute the following three steps:

Step 1: Transaction Design (Input)

The first step of a batch design is to specify the input or inputs that are to be the driving force of the batch program. The input driving force is called a transaction. In its simplest form, a transaction can be serial input from one or more files or databases (for example, sequential or VSAM files, DL/I, or SQL databases). Other simple transaction input sources are IMS/DC message queues, interfaces to other programs, and JCL parms.

When processing simple transaction input, you can implement reusable transaction input by reconfiguring the system after testing. Do this by changing the simple transaction input to a program-subprogram interface. Create a driving program performing the transaction input and remove the transaction input from the existing processing programs. Convert the processing programs to subroutines, and the transaction input is now reusable.

You can define more complex transactions for synchronizing multiple inputs. *Synchronizing* or *matching* the inputs depends on identifying the common input characteristics. For example, assume that all inputs contain an employee identifier. When *reading* the inputs, synchronization (matching) should occur for the employee name (part of identifier) of each input. The transaction in this case is a record area (or areas) that is to drive one iteration of the batch process and usually generate one or more lines of report output.

If input synchronization requires a *master* and a *transaction* to process against the *master*, use the CA Telon match feature. If synchronization requires any number of inputs, use the CA Telon merge feature. If input synchronization requires sorting before processing, use the CA Telon mainline sort or user sort features. For further information, refer to the *Data Administration* manual or *Design Facility Reference* for a description of the data access features of CA Telon you can use for transaction input processing.

Note: You *must* design the transaction input early in the batch definition phase because the structure (or resultant program) is based on the CA Telon transaction feature selected. Simple transaction input, match, merge, and mainline sort processing each result in a different generated program structure. For more information on batch program structures, see Chapter 7, "Program Hierarchical Structure."

Step 2: Database/Data Set Definition

The second step of a batch design enables you to identify the database(s) or data set(s) to be accessed by the program. You identify their characteristics and the type of access (for example, read, write, update) using the TDF User Exec definition screens (just as for online systems). CA Telon then performs the access from custom code.

Step 3: Processing Logic (Process)

The third step adds the processing logic to control execution and perform database or file access.

In this case, the program logical structure is simply:

- INIT—Initialization
- INPUT—The transaction access
- PROCESS—Custom code functional logic
- TERM—Termination processing

Note: Since no report writing stage exists, the logic simply includes the processing of each transaction.

Although there are multiple points where you can add processing logic, most custom code is added at the PROCESS logical point. You can incorporate user sort processing at any logic point in the routine. To sort a file before processing the file, implement a user sort in INIT processing.

CA Telon Partitioning

As indicated above, CA Telon emphasizes breaking up large systems into smaller, isolated modules for definition, programming, and testing and then, in a later step, integrating those components into a unified whole. Though not required, this decomposition technique lets you focus on simpler components during the development process and provides additional flexibility in the packaging of components.

Using this technique, you do not need to understand exactly how the components are to be integrated in the end. Usually, you will have a general idea, but not one where all the details have been defined. This allows a form of modeling to be used, where you create a batch definition (for instance to write a report) and execute it against sample data. You review the results, modify the batch definition, and rerun the program, even though the simple transaction file used in the model is to be replaced later by a more complex interface (for example, match/merge process).

When using this decomposition technique, you need to determine the unit for which you are creating a batch definition. Some of the characteristics of a unit are listed below :

- **Single report**—A batch definition cannot include separate reports. If two reports (printed output with different page numbers, formats, control breaks, and so on) are to be created, two batch definitions are required.
- **Sequential transaction input**—The transaction input must be considered to be accessed in proper sorted order, regardless if it is a file, database, or program interface. Note that we are considering here the logical order of transactions presented to the process and output stages, which can be created by properly passing the records through the program interface from a driving module or by sorting the records internally on initialization of this program.
- **Input/Process/Output structure**—The logical structure of the function to be executed must have the transaction input, process, and optional report output structure. If the integrated function involves the sequential processing of multiple files or databases asynchronously (one after the other, without concatenation), each is probably best defined as a single batch definition.

Note: A batch definition always results in the generation of a COBOL or PL/I program. That program can be the only program in the batch step to be executed. It can be the main processing routine of an integrated system with subroutines created from other batch definitions, or it can be a subroutine to be executed from another CA Telon or non-CA Telon program.

Sample Program

To illustrate the use of the above techniques, the following sample program is designed in two ways.

The program to be created is to perform the following three tasks:

1. Read a set of unordered selection criteria in a table. This criteria determines which database records are to be selected for updating.
2. Sequentially retrieve all segments of a particular type from an IMS database.
3. Based on the selection criteria, update a second database with the information from the first and write the information out to a report.

The first design creates a single batch definition as follows:

1. The segment report (item 3) would be defined as the prime output.
2. The database segments to be retrieved sequentially (item 2) would be defined as transaction input (the driving force).
3. The *validation* of each transaction segment against the selection criteria table and the updating of the second database (item 3) would be special processing.
4. The loading of the selection criteria table (item 1) would in this case be *initialization* prior to the main task.

A second design might be used if the creation of the selection table was a more complex task, with possibly its own report. In this case, two batch definitions could be created. One would include:

1. The selection table report as prime output.
2. The reading of the selection criteria as the transaction input.
3. The placing of acceptable criteria into the table as special processing. At the end of all transaction iterations (when the table is complete), the second batch definition would be called, passing the completed selection criteria table.

The second batch definition would be the same as in the first design case above except that the fourth step, table initialization, would not be done.

Note that the second design is only justifiable if the creation of the selection criteria table is particularly complex in itself. In that case, it is better to treat it as a self-contained unit.

Note also in this case that if the main maintenance task and report needed to be modeled before the selection criteria was implemented, it would be easy to create the second batch definition with a simple table initialization step that could be later removed.

The batch definition modeled could also be easily modified to be included in a system where the sequential database is read by another program and each record passed to this program for processing.

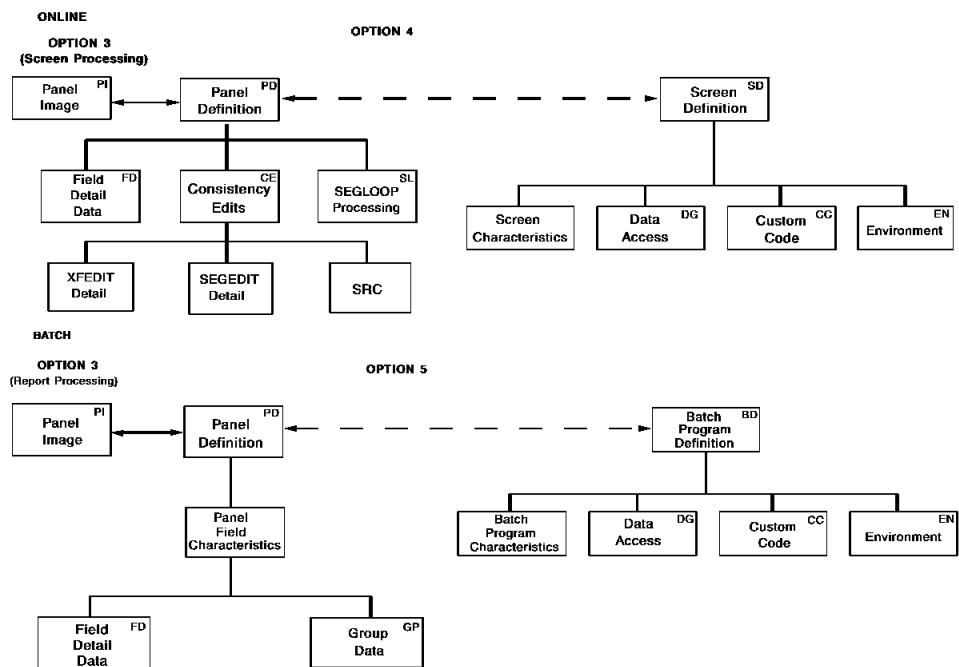
Chapter 3: CA Telon Design Facility

This chapter details CA Telon program structure and introduces you to the CA Telon Design Facility (TDF). This chapter also contains charts describing the flow of the various TDF screens and describes the TDF Main menu.

It is important that you understand the information contained in this chapter before reading the remainder of this guide.

TDF Program Structure

The following diagram illustrates the logical sections of a program developed using the TDF. Read the diagram from left-to-right and from top-to-bottom.



The following is a basic description of each component of the structure.

Step 1

Panel image

This is the layout of the screen or report. You create the panel image by keying (*painting*) the literals and variable fields on the screen in the format that you want them to appear.

Step 2

Panel definition

This specifies all the characteristics of the panel as defined by the subcomponents below it.

Panel field characteristics (field data detail)

This defines the basic characteristics of each field defined on the panel image. This includes identifying the source of displayed data, the destination of entered data, editing, additional attributed settings, and so on.

Consistency edits

This specifies each consistency edit defined by means of a cross-field edit (XFEDIT), segment edit (SEGEDIT), or source code edit (SRC). Note that you can add additional consistency edits using custom code when the screen definition is created. Consistency edits defined through the XFEDITS/SEGEDITS/SRC parameters are tied to the panel definition, while those defined through custom code are tied to the screen definition.

SEGLOOP Processing

This defines the use and characteristics of loop processing on a list program.

Step 3

Screen/report definition

This is a representation of the total definition, the input to the CA Telon Generator.

Screen/report characteristics

This defines some basic characteristics about the screen or report, such as the transfer work area that you want to use in this screen/report definition, the PF keys that you want to activate, and so on.

Data Access

This defines whether you want to access databases, VSAM data sets, or sequential data sets.

Automatic Exec

This defines the type of access (that is, read on output, insert on input, and so on).

User Exec

This defines I/O that is performed from custom code added to the definition. CA Telon requires user exec I/O when the access to a segment differs from execution to execution of a program. For example, if it is sometimes read and sometimes updated.

Custom code

This defines PL/I or COBOL custom code that you want to insert into the generated program at various logical points. Custom code enhances CA Telon programs by providing them with the ability to produce any program that you can produce with PL/I or COBOL.

Environment

This specifies the environment in which CA Telon generates the program.

Note: At this point, you do not have a detailed knowledge of the components listed above. This discussion simply familiarizes you with the CA Telon components. Any questions that you have should be answered in later chapters of this guide.

Final CA Telon Program

The final CA Telon-created COBOL or PL/I program conforms to ANSI COBOL or OS PL/I standards in all ways. However, when you use the TDF, you must think of your program in terms of three basic program units:

- A panel image
- A panel definition

- A screen definition, report definition, driver, batch definition, nonterminal definition, or stored procedure (these are collectively termed *program definition*). The CA Telon Generator uses these three units to make one CA Telon source code file for your program. Within the TDF, the program components are named as follows:
 - Panel image—screen-name.PI
 - Panel definition—screen-name.PD
 - Screen definition—screen-name.SD
 - Report definition—report-name.RD
 - Driver definition—driver-name.DR
 - Batch definition—batch-name.BD
 - Nonterminal definition—nonterm-name.ND
 - Stored Procedure definition—stored name.SP

The above names represent a **HEADER** plus an **ID**, where the **HEADER** and **ID** add up to a total of five or six characters (depending on whether the header is one or two characters long). For example, a panel definition could have **HEADER** and **ID**: TRFRED.PD.

Where:

- TR is the **HEADER**
- FRED is the **ID**
- PD identifies a **panel definition**

The three basic sections of a program each have specific tasks:

1. Panel image—The panel image is the most basic unit of your program. It is a layout of a screen or a report. You create it by keying in (*painting*) the literal and variable fields onto the screen in the format you want the final screen or report to appear. CA Telon uses the panel image to capture field locations, field types, and field lengths.
2. Panel definition—The panel definition describes all specifications relative to the fields that are mapped to and from the panel image.

The panel definition includes the following:

- Panel image—This is described under item 1 above.
- Field edits—These edits define the basic characteristics of each field included in the panel image. They include items such as field mapping, field editing, and attribute settings.
- Consistency edits—These edits (XFEDIT, SEGEDIT, and SRC) check incoming fields against each other and against existing database or data file data.
- SEGLOOP processing—This defines how the panel definition displays and reads lists of fields.

Note: Batch programs do not require a panel image or panel definition.

3. Screen, report, driver, batch, nonterminal, and stored procedure definitions—Each of these is the basic unit of a TDF designed program. They each include the panel definition and all other specifications necessary to export your CA Telon source code to the CA Telon Generator for compilation into a completed COBOL or PL/I program.

The definition includes the following:

- Panel image—This is described under item 1 above.
- Panel definition—This is described under item 2 above.
- Screen, report, driver, batch, nonterminal and stored procedure characteristics—These include the basic characteristics of your screen, report, driver, batch, nonterminal, or stored procedure program. These include items such as the name of the transfer work area and the names of any custom code modules.
- Data access—This defines the databases and data files that the programs access. It includes those automatic exec I/Os that are the same for all executions of the screen, report, driver, batch, or nonterminal definition. It also includes any user exec I/O that your custom code performs. You must use the user exec I/O when your I/Os vary from one execution to the next based on specific conditions.
- Custom code—This is any COBOL or PL/I code you want to include in your program to perform logic beyond that which CA Telon normally performs.

Programming with the TDF

Use the following steps when entering your program into the TDF:

1. Create the panel image
2. Create the panel definition
3. Create the screen, report, driver, batch, or nonterminal definition (these are collectively termed as *program definition*)

This does not prevent you from going back and re-entering or changing items. It simply suggests a smooth order to enter your program.

The following topics provide you with details on how to do this.

Note: Do **not** include single quotes (') or ampersands (&). within any value within the TDF except in the following cases. Single quotes (') are allowed in only the beginning and end of the value if the value has embedded spaces or if two single quotes (") or ampersands (&&). are used within a value. If you do not follow these rules, errors will occur during program generation.

Online TDF Program Structure

This subject provides an overview of the panel image, panel definition, and program definition for online users. If you are a batch or CICS Nonterminal user, please skip this subject and go to the subject titled "Batch TDF Program Structure" later in this chapter. A nonterminal program uses a batch program structure.

You create an online program by following these steps:

1. Create a panel image
2. Create a panel definition
3. Create a program definition

Creating a Panel Image

As introduced in Chapter 1, "General Information", the panel image is the layout of the image of your screen. You create it by keying (*painting*) the literal and variable fields onto the Edit Panel Image screen. CA Telon, in turn, uses this image to capture literals, field locations, field types, and field lengths.

Paint an Image on the Edit Panel Image screen by using literals, output fields, input fields, outin fields, select fields, and the literal break character. The images that you can place on the Edit Panel Image screen are termed *field types*.

Creating a Panel Definition

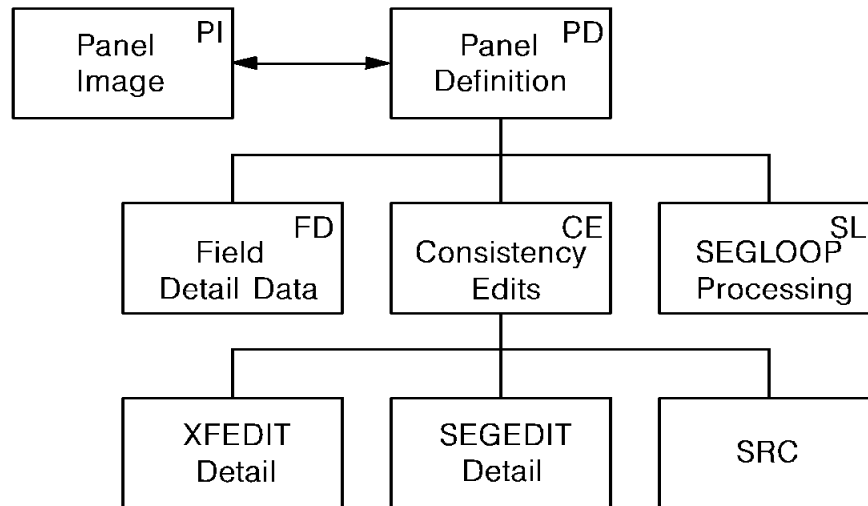
After completing the panel image, your next task is to create a panel definition. The panel definition defines all fields mapped to and from the terminal for a screen definition, or to a printer for a report definition. The panel definition provides data, such as:

- The source of data being mapped to an output field
- The source that the data from an input field will be mapped to
- Whether the application user must enter a particular field
- What edit tests a field must pass
- How CA Telon reformats data during input
- The special attribute characters that you want to use

The Panel Definition Structure diagram shows the portion of your TDF-generated program specified in the panel definition. Notice that the panel definition includes the panel image.

The field characteristics are the first items you must enter into the panel definition. The field characteristics include field mapping, editing, additional attribute settings, and so on. After you specify the field information, specify consistency edits to be included for the panel definition. Consistency edits can be crossfield validations (the comparison of two or more variable fields), database validations (the checking for the existence or nonexistence of a segment on the database), or SRC edits (the insertion of COBOL or PL/I statements).

If the panel definition includes the listing of multiple groups of the same type of data, from either the reading of multiple segments or an array, define a SEGLOOP definition for the panel definition. For detailed information, see Chapter 6, "Creating the Panel Definition."

Panel definition structure**Creating a Program Definition**

After creating your panel image and panel definition, the next step in the program creation process is to create a program definition. That definition can be one of the following:

- Screen definition
- Driver definition
- IMS/DC report definition
- CICS nonterminal definition
- Batch definition
- Stored Procedure definition

The Program Definition Facility allows you to do the following:

- Define general program characteristics
- Define data that the program accesses
- Create procedural custom code
- Identify the environment in which the program operates

The TDF Program Definition Structure diagram shows the portion of your TDF-generated program specified in the program definition. Notice that the program definition includes the panel image and panel definition.

To create a program definition, you must specify screen/report characteristics. The screen/report characteristics include cursor location, name of the transfer work area, and names of custom code used at various portions of the program.

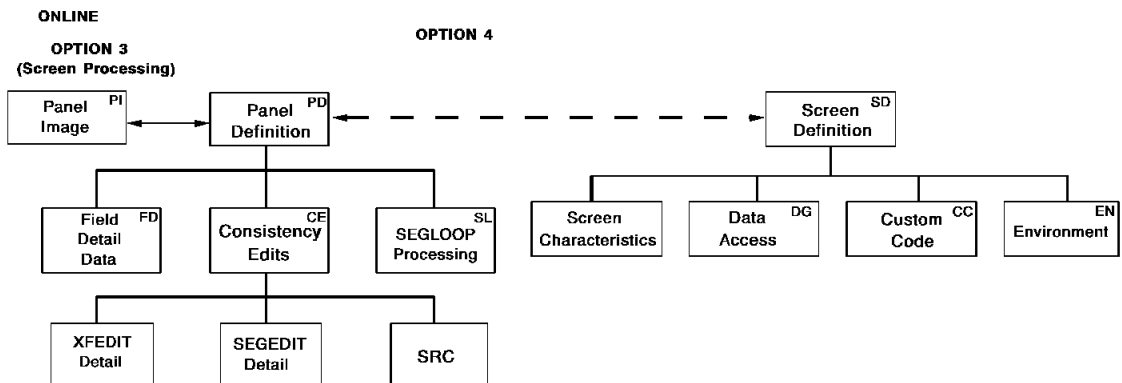
You must also specify the type of data access that you want to use. You can define two types of data access: automatic (auto) exec and user exec. When your program always does the same action with your data (read, write, create, or update), you can create auto exec I/O. The CA Telon-generated program then accesses that data function with each execution of the program.

When the same I/O segment must perform different processing based on varying conditions in the program, you must create user exec I/O.

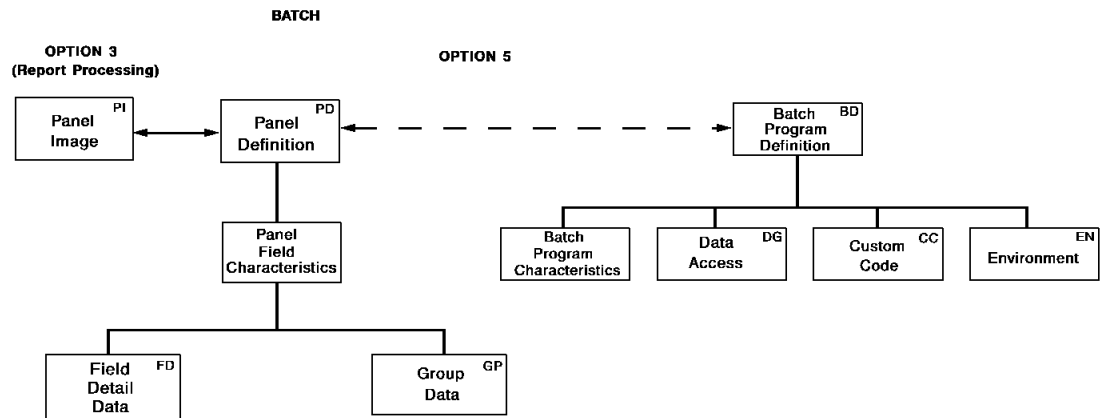
Although the TDF automatically generates most of the code needed in your program, there are some cases that require your own custom code to correctly complete the task you want to perform. You can insert this custom code into almost any point of your CA Telon-generated program.

The final step in creating a program definition is identifying the environment in which you want the program to run. For detailed information on how to create the program definition, see the chapter "Creating the Program Definition."

TDF Program Definition Structure



TDF Program Definition Structure—Batch



Batch TDF Program Structure

This subject provides an overview of the panel image, panel definition, and batch definition (for batch users). If you are an online user, please skip this subsection.

You create a batch report program by following these steps:

1. Create a panel image for report
2. Create a panel definition for report
3. Create a program definition

You create a batch program for which there is no report by following these steps:

1. Specify the transaction input
2. Specify the database(s)/data set(s) to be accessed by the program
3. Specify the processing logic of the program

Creating a Panel Image

The first step in creating a batch program that produces printed output is to create a panel image of that report. This image is created basically the same way as you create a panel image for a screen program, by way of the Panel Specification menu.

But there the similarity ends. When creating a panel image for a *screen* program, you lay out the screen as it will appear in the completed program. With a panel image for a *batch* program, you lay out *groups* of formats. Each group represents a particular event occurring on the report; for example, a page heading (TOPPAGE), the bottom of page detail (BOTPAGE), and so on. The CA Telon Batch Facility defines six different events that can occur in your report. The following list describes the six groups of lines defining these events. The Batch Report Events diagram illustrates where the events occur in the report.

Note: A batch program need not have a panel image if it is not a report. A batch program can be just a transaction or can just create another file.

Batch Report Events

TOPPAGE	{	BILL'S WIDGETS IOWA CITY, IOWA				
TOPDETAIL	{	NAME	ADDRESS	CITY	STATE	PAID
DETAIL	{	JOHN SMITH	1234 Green St.	Lost Nation	IA	100.00
"	{	—	—	—	—	—
"	{	—	—	—	—	—
CONTROL	{	The state is now changed. There were 123 names in IA and \$123.00 received.				
SUMMARY	{	The report is now done. There are.... 703 names in 218 cities. Total paid is \$14,444.44				
BOTPAGE	{	Page 8				Our Report

- **TOPPAGE**—The TOPPAGE group defines the data printed at the top of a physical page. It is the report heading.
- **TOPDTL**—The TOPDTL group defines the data printed at the top of a detail line. Usually this is the column headings for the detail lines.
- **DETAIL**—The DETAIL group defines the data printed in the main body of the report. This is the output of the report—the data driving the program. At least one DETAIL group must be present to produce output.
- **CONTROL**—The CONTROL group is data printed only when a specific data value has changed between printing the last detail line and the current line. It is the data printed on a control break. It is, in fact, the control break.
- **SUMMARY**—The SUMMARY group defines the data printed at the end of the report. It is the control break that occurs when there are no more detail lines to print. It can occur only once in a report.
- **BOTPAGE**—The BOTPAGE group defines the data printed at the bottom of a physical page. It is the page footer.

You can use all groups except SUMMARY several times in a single CA Telon batch program. You must code at least one DETAIL group in each program.

In addition, the batch program's panel image can only use *output* fields, *literal* fields, and the *literal break character*.

Each of these fields is designated by a particular character. Output characters cannot be used within literals. The default field designators are:

- Output—>
- Literal Break—\

For information on how to change these default values, see Chapter 4, "User Profile Maintenance."

The following screen illustrates an example of a *painted* panel image (using the line edit mode) with painted output and literal fields.

[illegible]

Note that the layout of a report page is not simply painted as it will exist when printed. This is because each page of a report is not necessarily formatted exactly the same as others and because a page usually includes groups of fields whose format is repeated and should not have to be painted redundantly. For detailed information on creating the panel image, see the chapter "Creating a Panel Image."

Creating a Panel Definition

After completing the panel image, your next task is to create a panel definition. The panel definition defines all fields mapped to a printer for a batch definition.

The panel definition provides data, such as:

- The source of data being mapped to an output field
- What edit tests a field must pass
- The special attribute characters that you want to use

The Panel Definition Structure (batch) diagram shows the portion of your TDF-generated program specified in the panel definition. Notice that the panel definition includes the panel image.

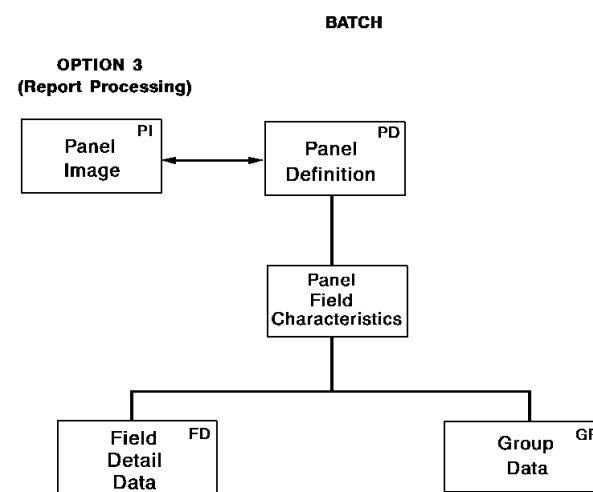
The field characteristics are the first items you must enter into a panel definition. The field characteristics include field mapping, editing, additional attribute settings, and so on.

The panel definition for CA Telon Batch and CICS nonterminal lists a field definition within a logical group. There are six types of CA Telon groups:

- TOPPAGE
- TOPDTL
- DETAIL
- CONTROL
- SUMMARY
- BOTPAGE

The user decides which group a field will be associated with upon processing. For further information on batch and CICS nonterminal groups and creating the panel definition, refer to the *Design Facility Reference*.

Panel definition structure-batch



Creating a Program Definition

After creating your panel image and panel definition, the next step in the program creation process is to create a program definition. This program definition consists of a batch definition.

The batch definition facility allows you to:

- Define general program characteristics
- Define data that the program accesses
- Create procedural custom code
- Identify the environment in which the program operates

The TDF Program Definition Structure (Batch) shows the portion of the TDF-generated program specified in the program definition. Notice that the program definition includes the panel image and panel definition.

To create a program definition, you must specify batch characteristics. Batch characteristics include items such as the report destination (for output producing programs), the name of custom code used at various portions of the program, and so on.

You must also specify the type of data access that you want to use. You can define two types of data access: automatic (auto) exec I/O and user exec I/O. When your program always does the same action with your data (read, write, create, or update), you can create an auto exec I/O. The CA Telon-generated program then accesses that data function with each execution of the program.

When the same I/O segment must do different processing based on varying conditions in the program, you must create user exec I/O.

Although the TDF automatically generates most of the code needed in your program, there are some cases that require your own custom code to correctly complete the task you want done. You can insert this custom code into almost any point of your CA Telon generated program.

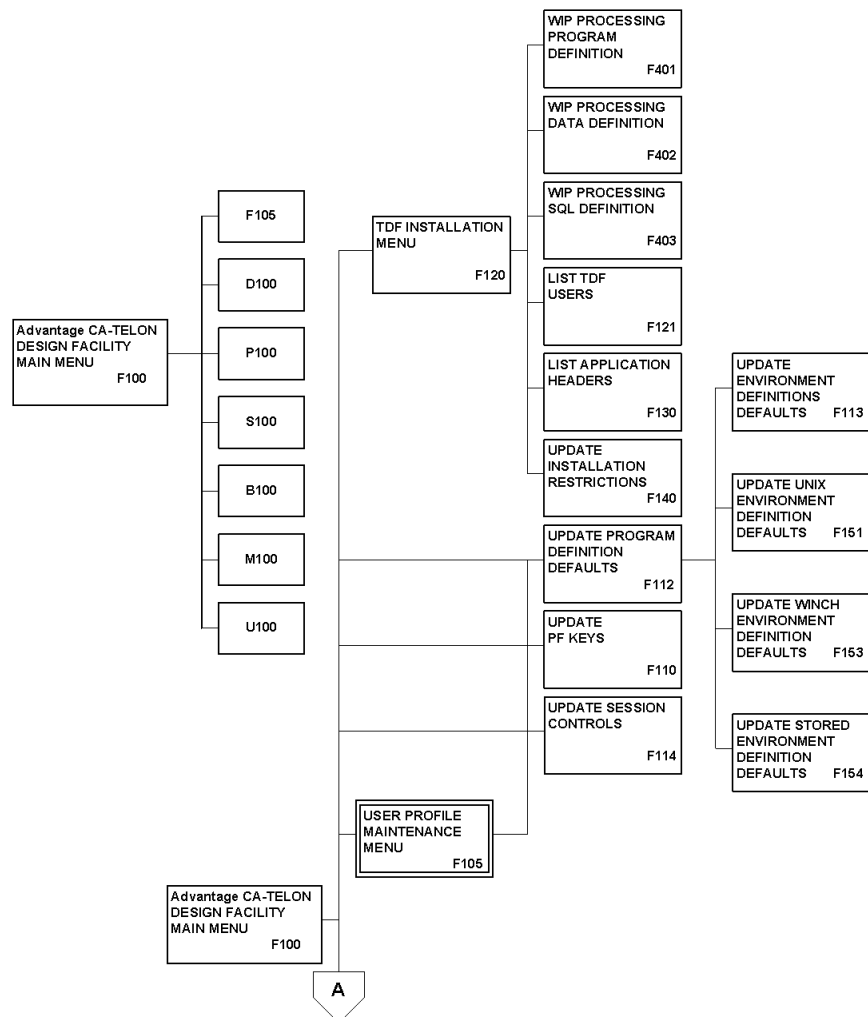
The final step in creating a program definition is identifying the environment in which you want the program to run. For information on how to create a batch program definition, see the chapter "Creating the Program Definition."

TDF Screen Organization

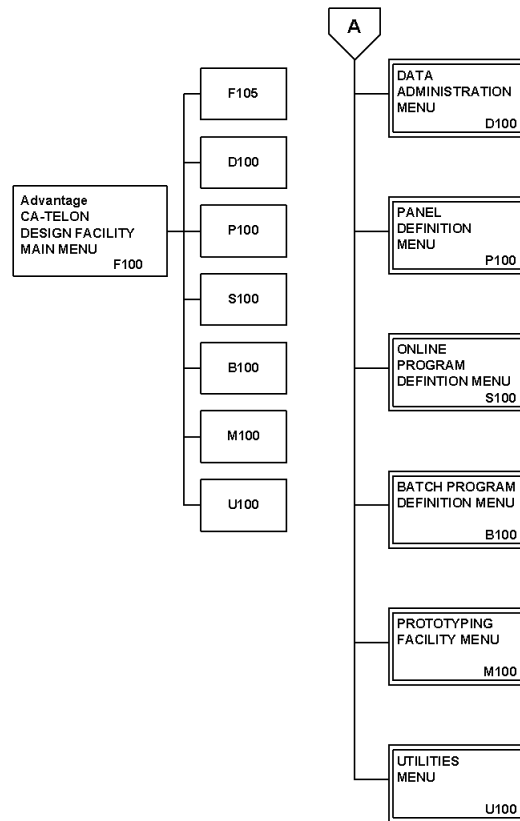
The CA Telon Design Facility (TDF) is organized by panel or screen flow. Each screen represents a logical step in program creation. The creation of programs may require the creation of many screens. The following flowcharts illustrate the TDF screen organization and the logical flow between screens.

Note: Specific groups of screens are used to create the panel image, panel definition, and program definition.

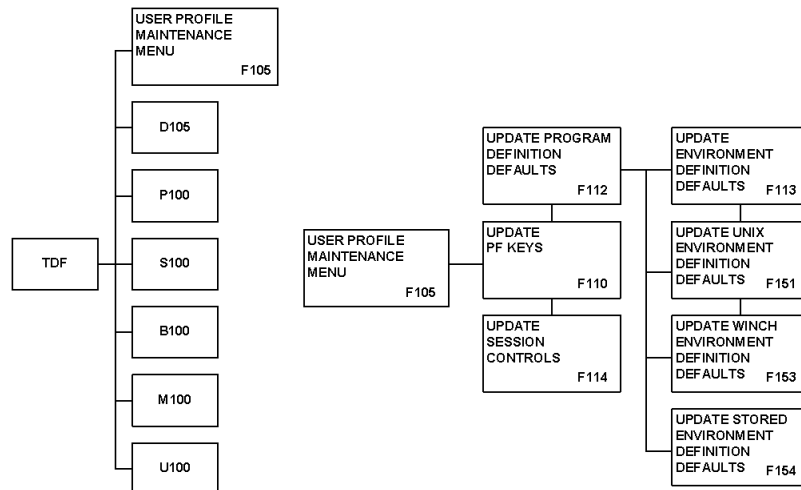
TDF Main Menu Screen Organization



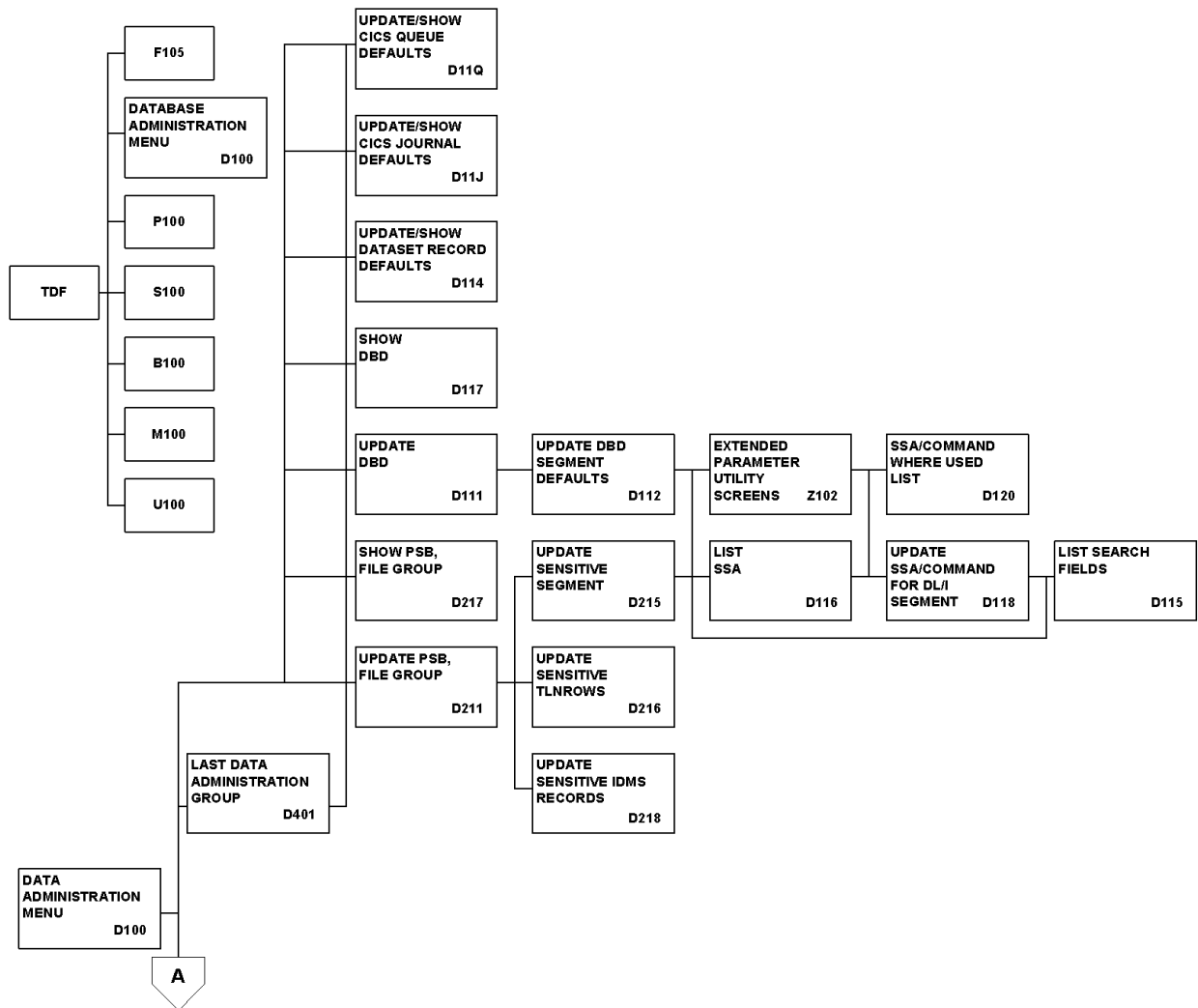
TDF Main Menu Screen Organization (continued)



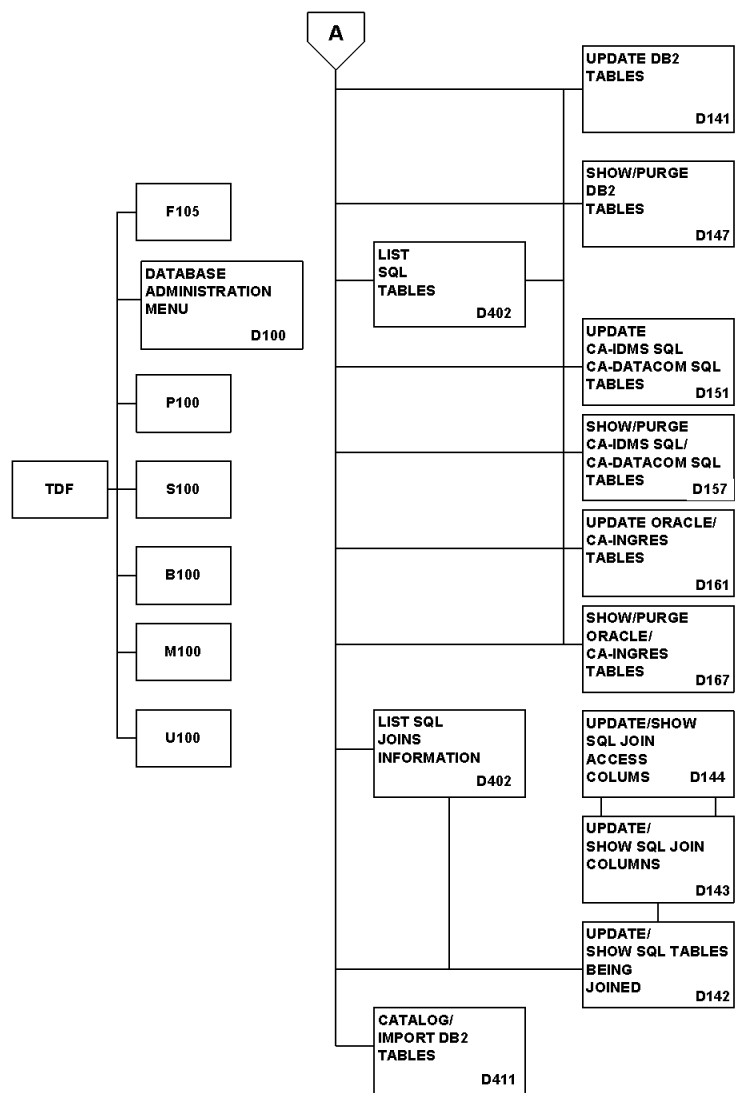
TDF Option 1 Screen Organization



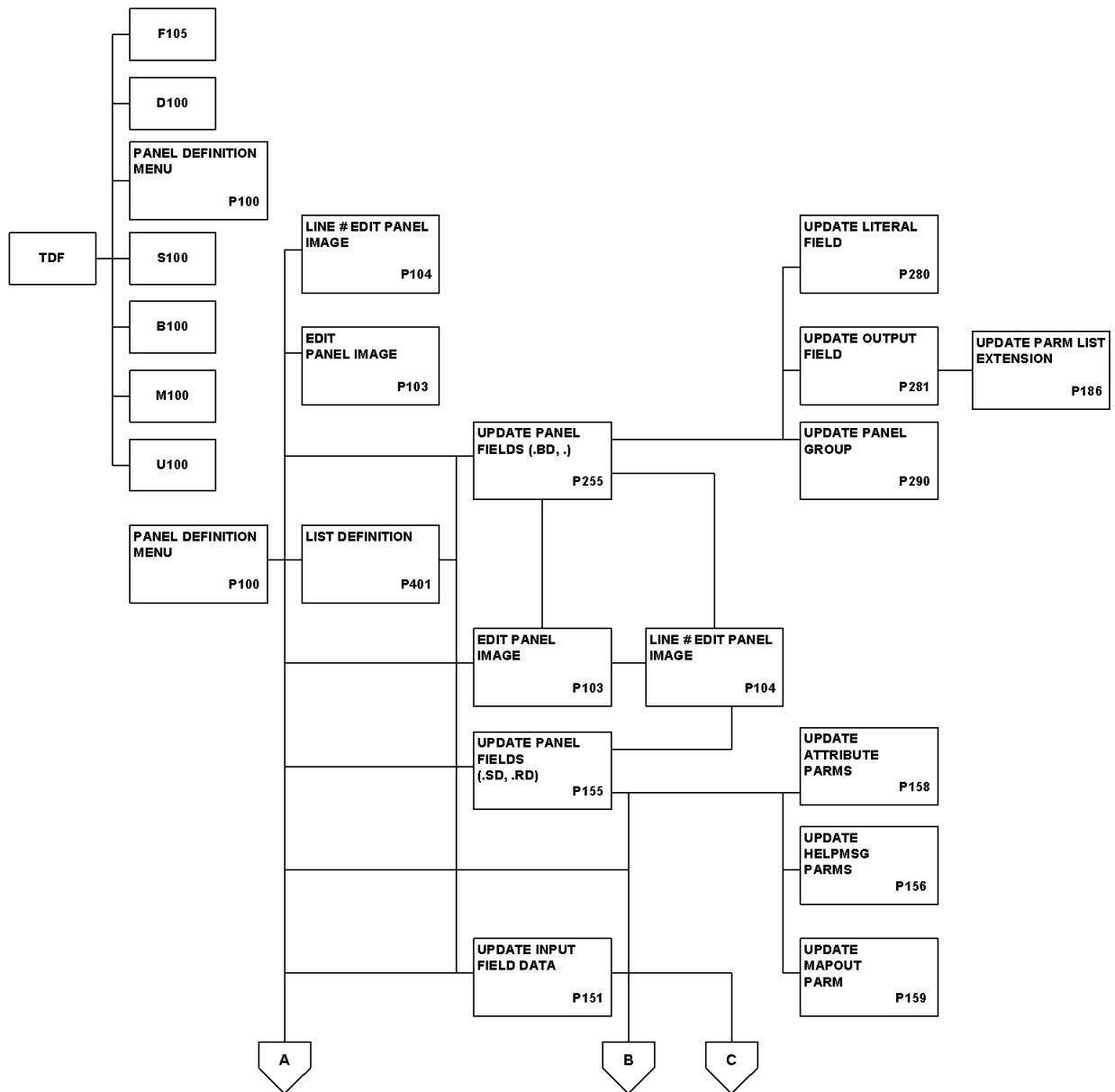
TDF Option 2 Screen Organization



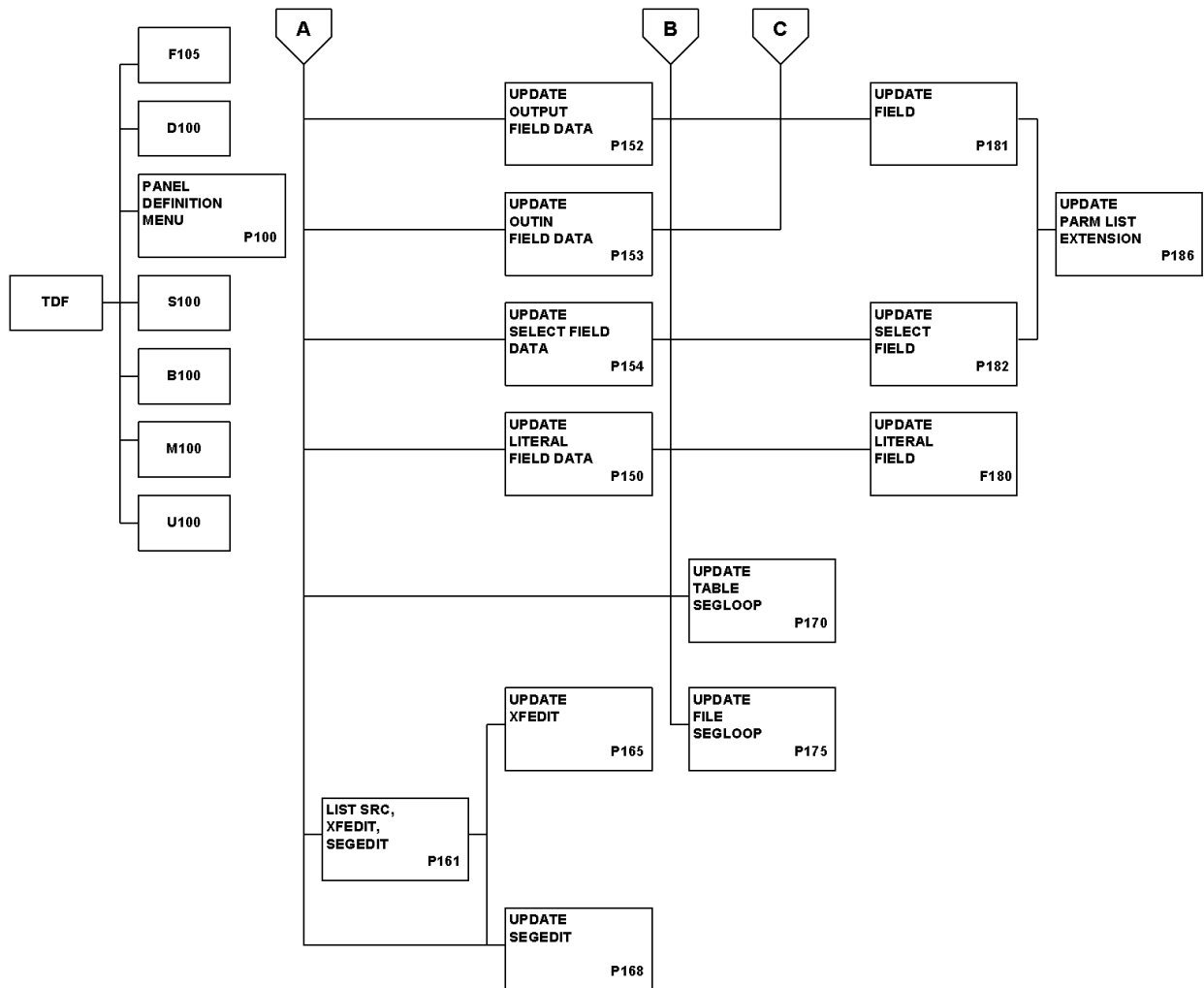
TDF Option 2 Screen Organization (continued)



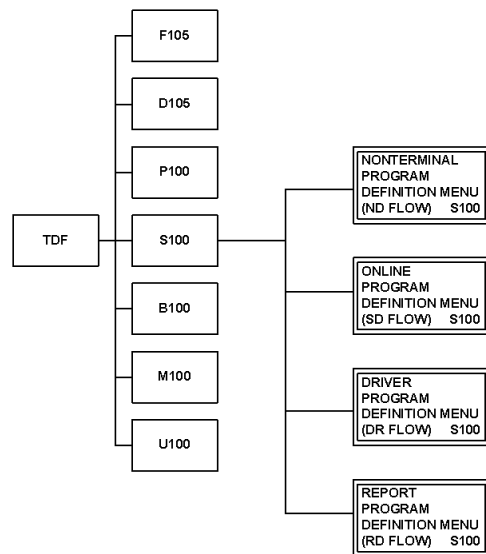
TDF Option 3 Screen Organization



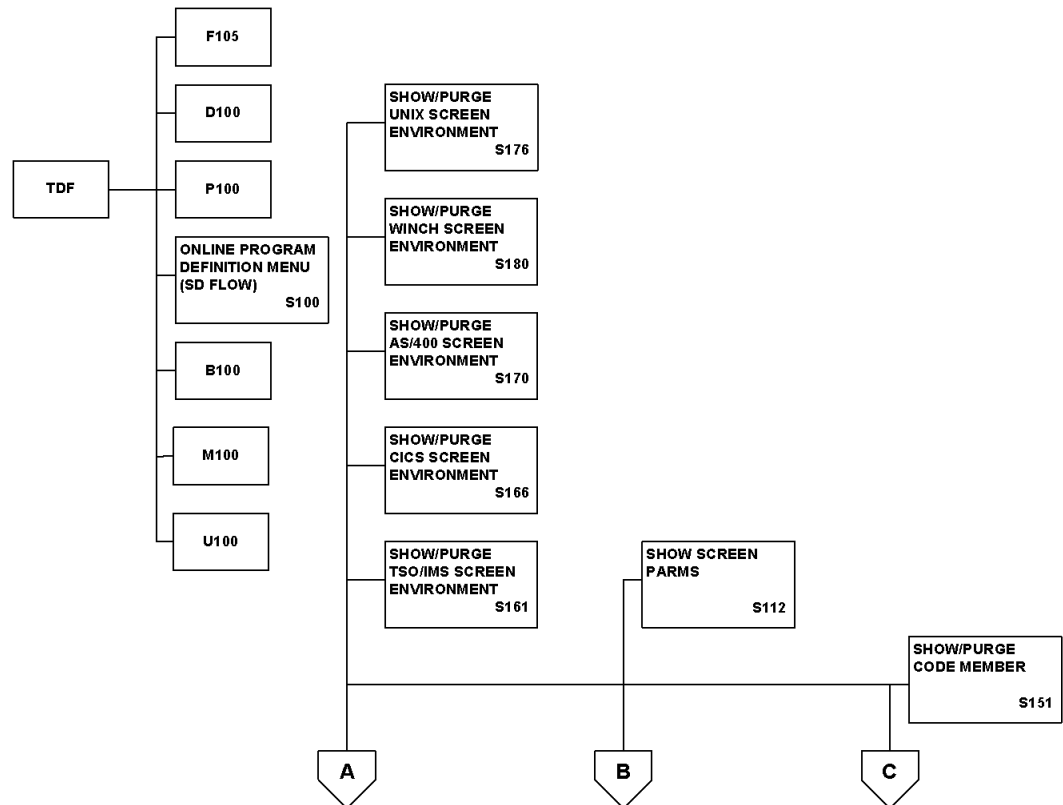
TDF Option 3 Screen Organization (continued)



TDF Option 4 Screen Organization



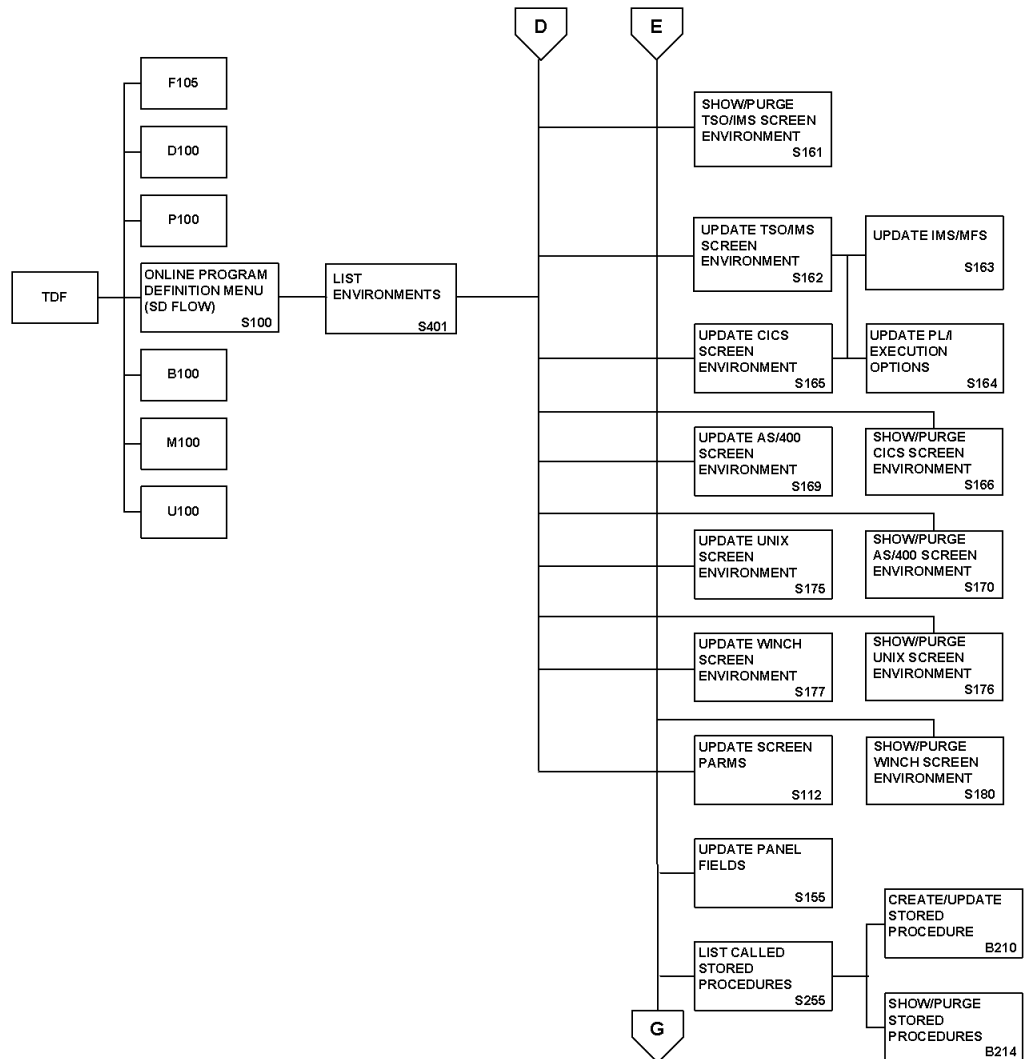
TDF Option 4 Screen Organization (continued)



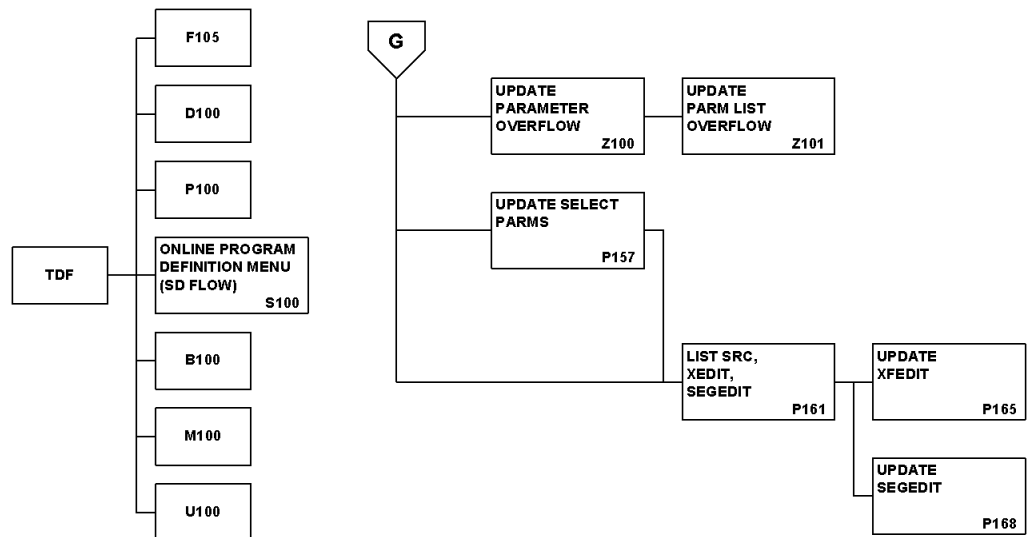
```

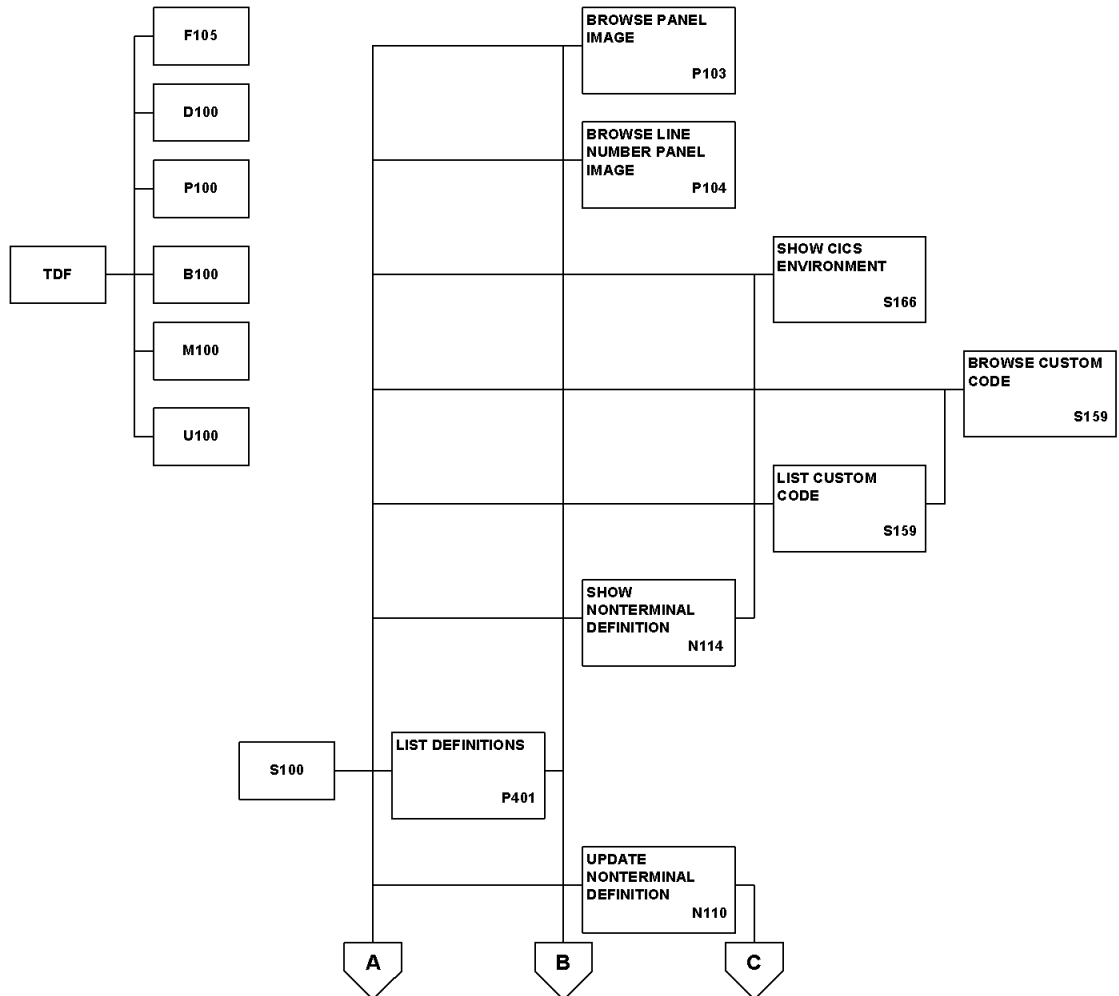
graph TD
    TDF[TDF] --- OPDM[ONLINE PROGRAM DEFINITION MENU  
(SD FLOW) S100]
    OPDM --- F105[F105]
    OPDM --- D100[D100]
    OPDM --- P100[P100]
    OPDM --- B100[B100]
    OPDM --- M100[M100]
    OPDM --- U100[U100]
    OPDM --- LD[LIST DEFINITIONS P401]
    LD --- SP[SHOW/PURGE SCREEN DEFINITION S114]
    LD --- US[UPDATE SCREEN DEFINITION S110]
    US --- US2[UPDATE SCREEN DEFINITION S110]
    US --- LCC[LIST CUSTOM CODE S159]
    US --- UGD[UPDATE GROUP DATA S125]
    LCC --- ECM[EDIT CUSTOM CODE MEMBER S151]
  
```

TDF Option 4 Screen Organization (continued)

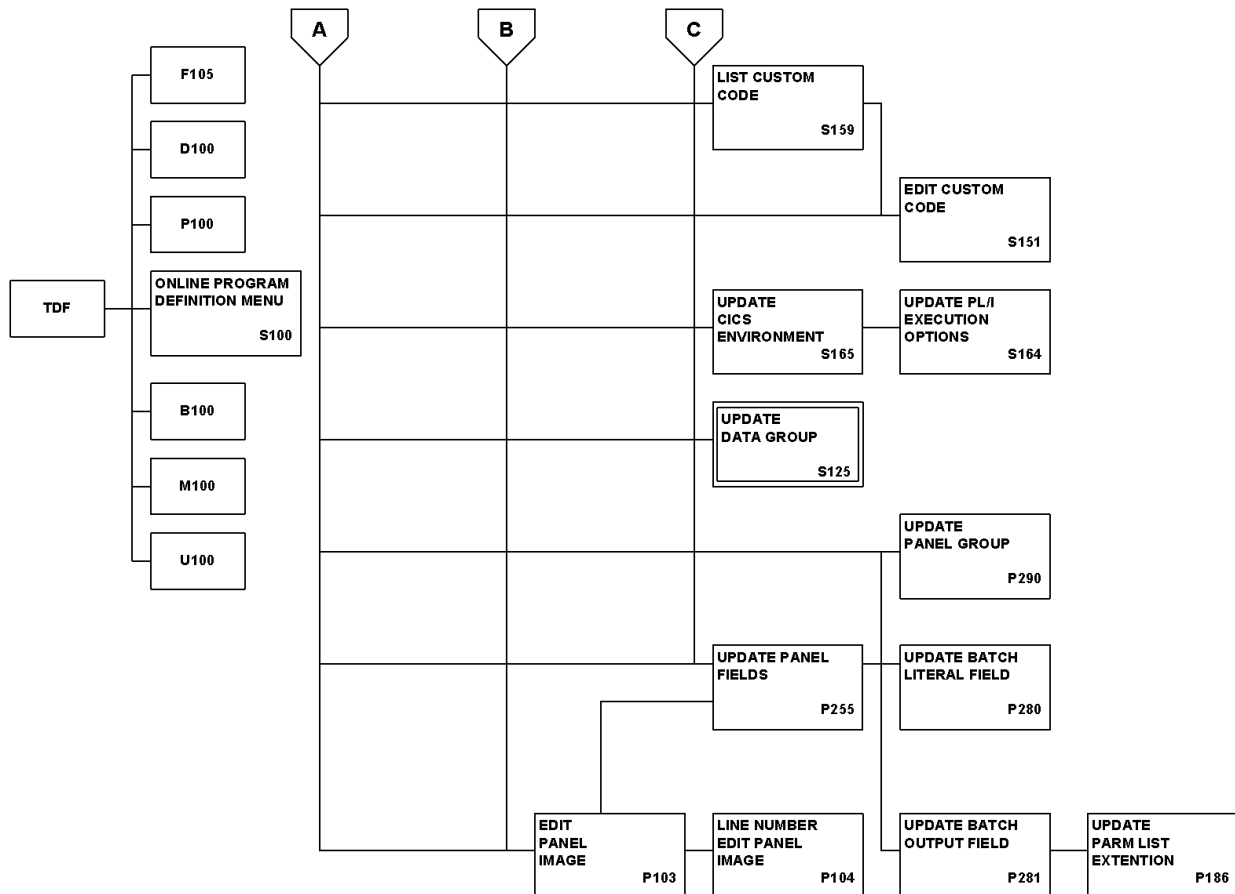


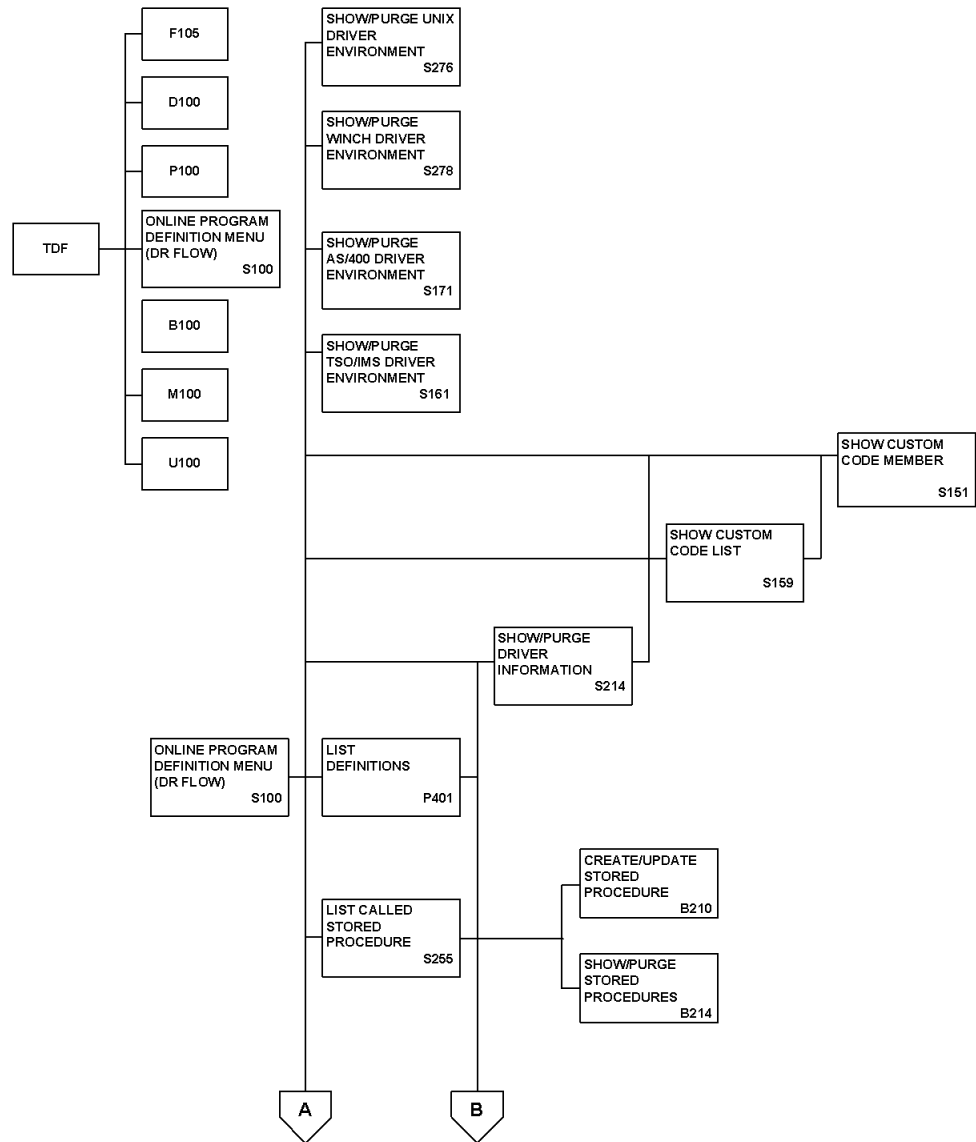
TDF Option 4 Screen Organization (continued)



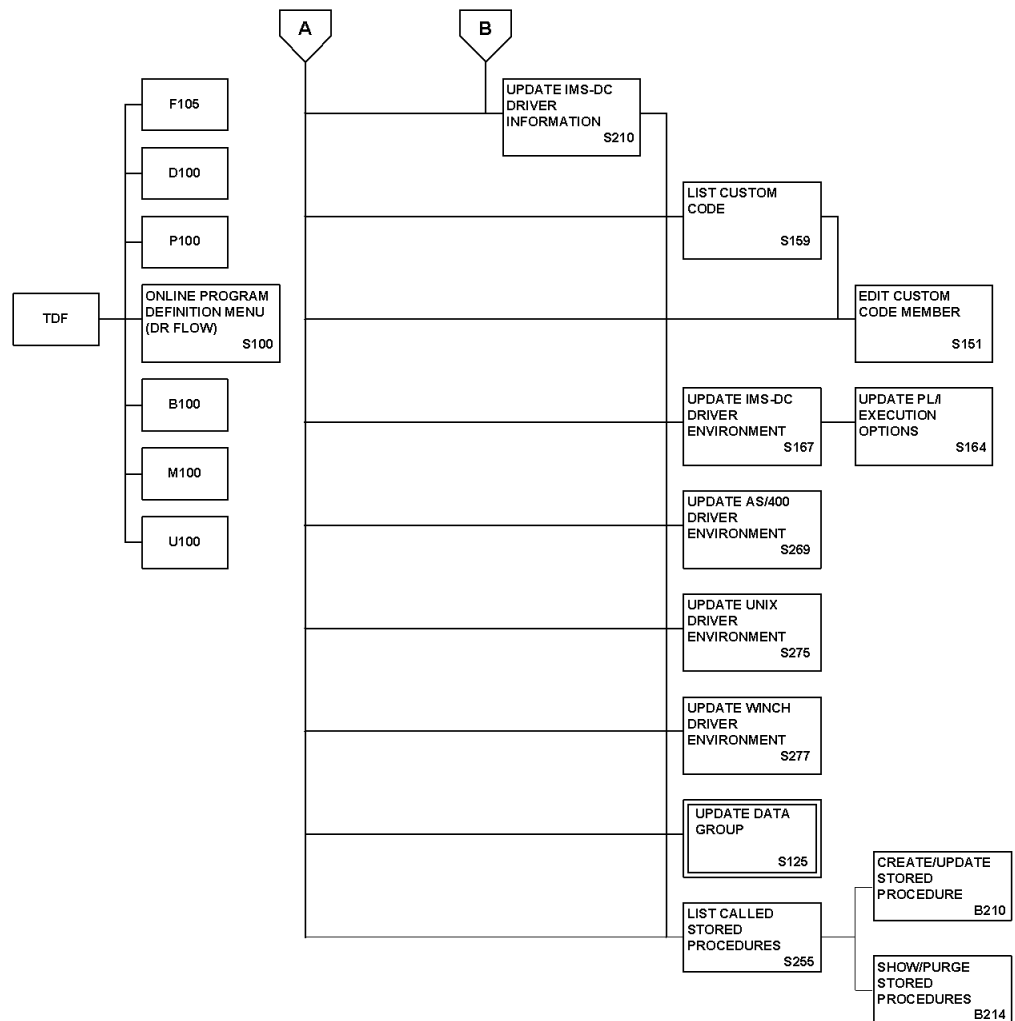
TDF Option 4 Screen Organization (continued)

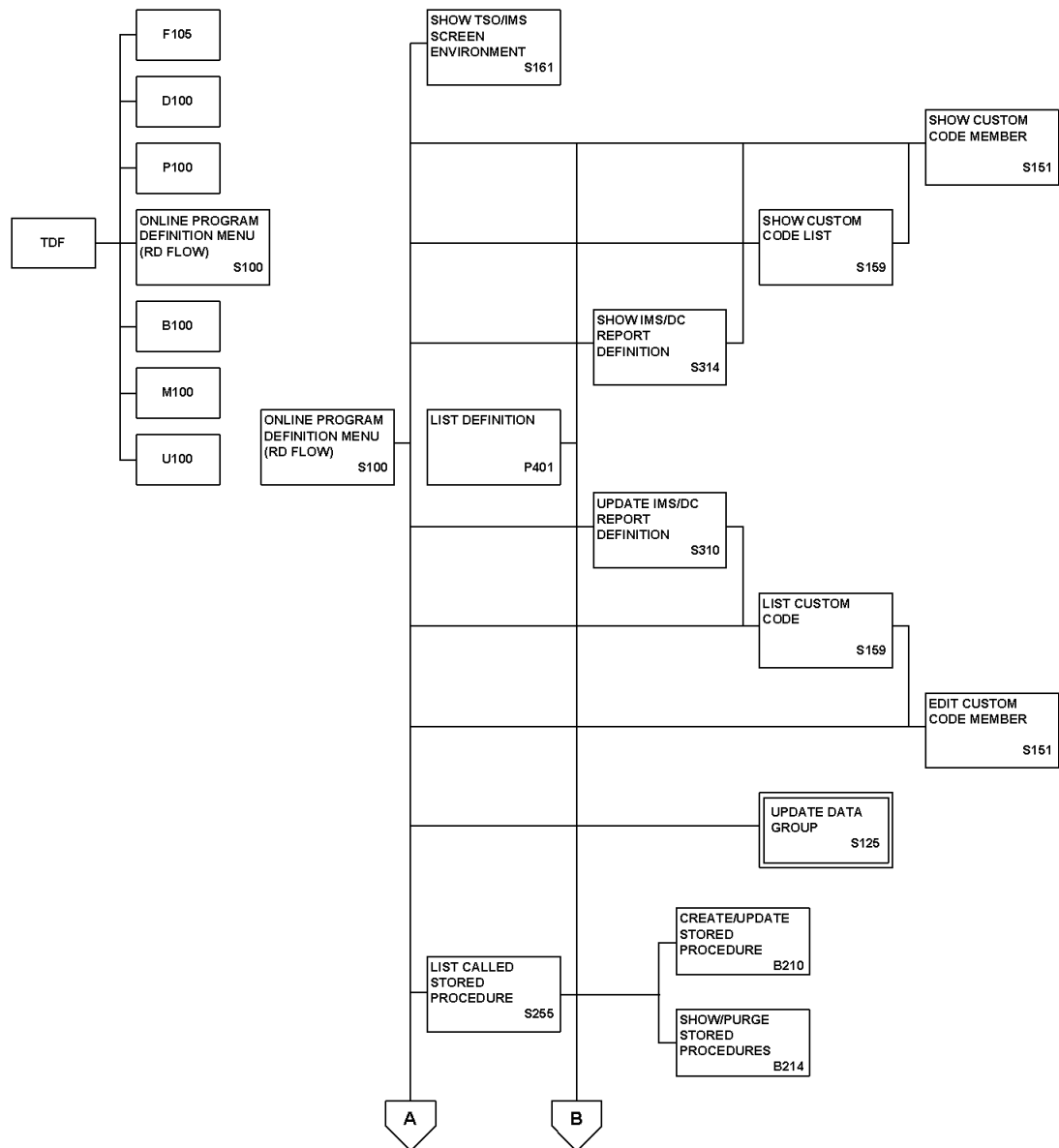
A



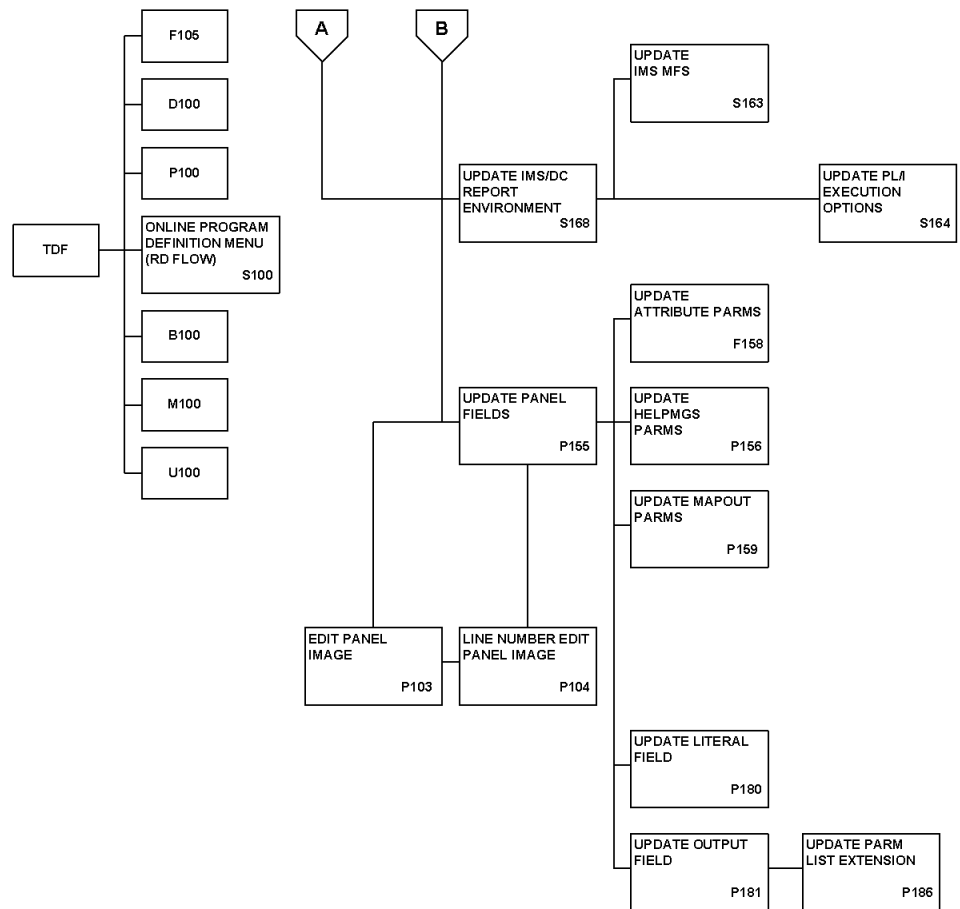
TDF Option 4 Screen Organization (continued)

TDF Option 4 Screen Organization (continued)

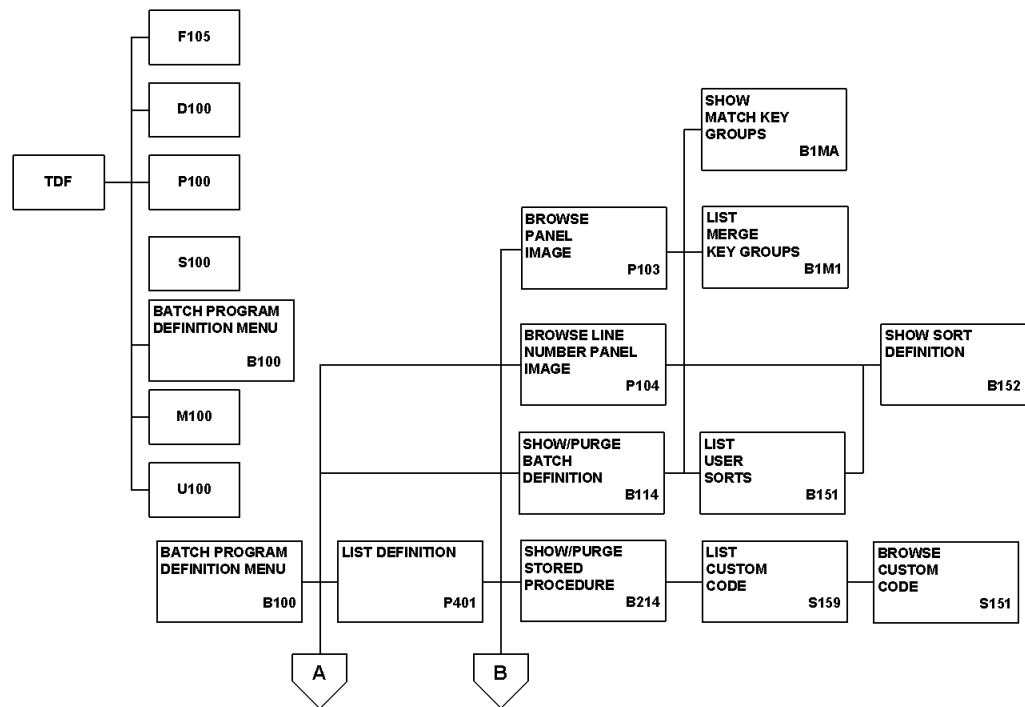


TDF Option 4 Screen Organization (continued)

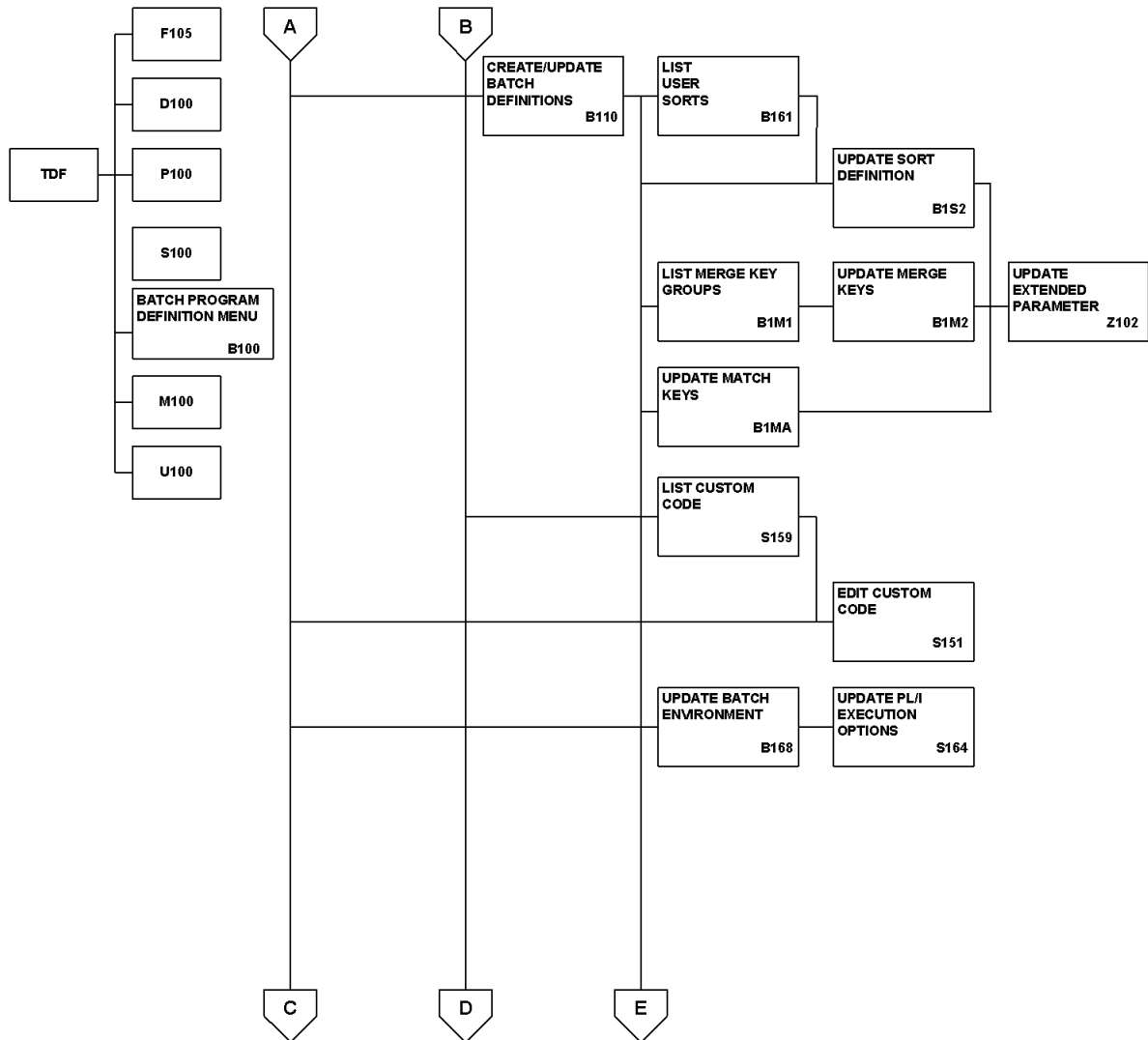
TDF Option 4 Screen Organization (continued)



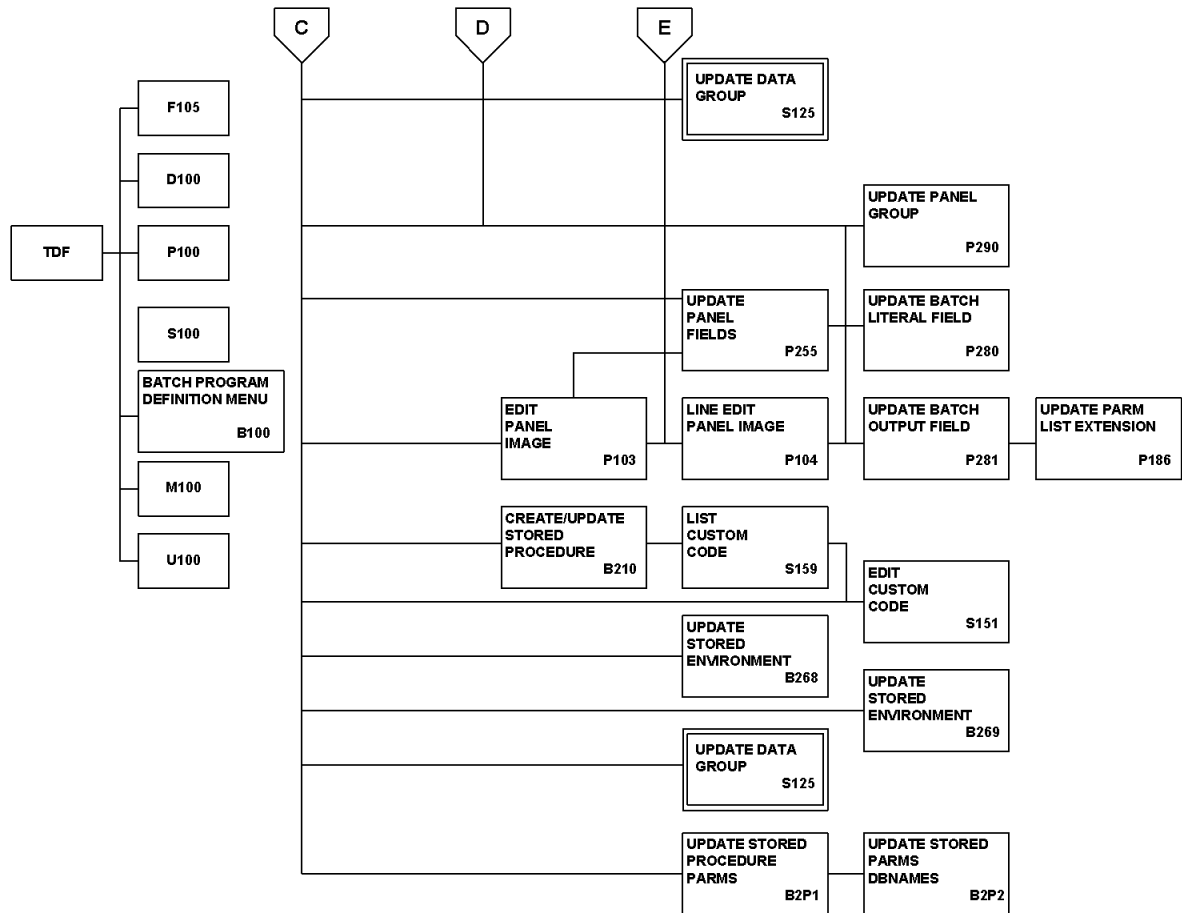
TDF Option 5 Screen Organization



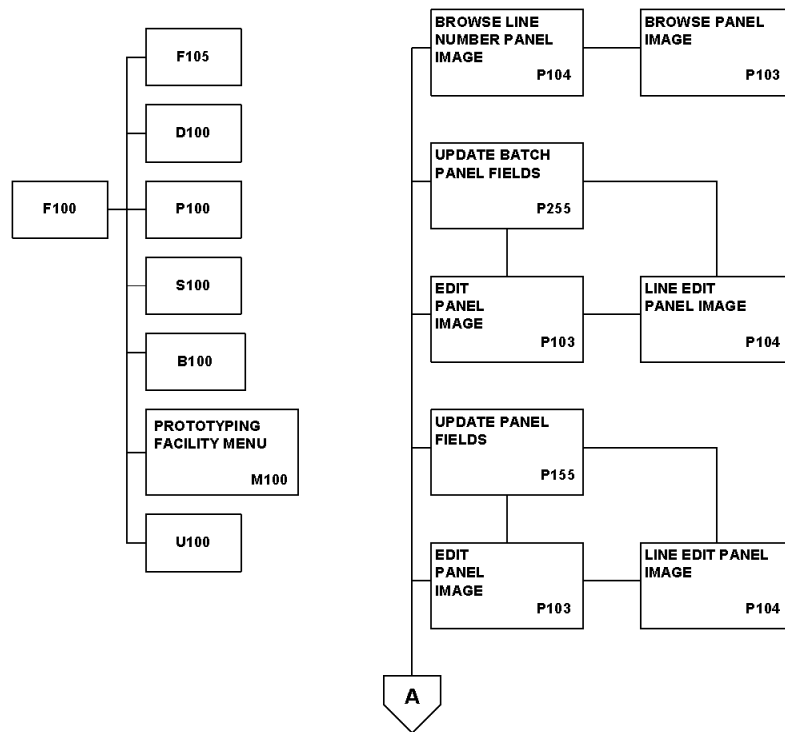
TDF Option 5 Screen Organization (continued)



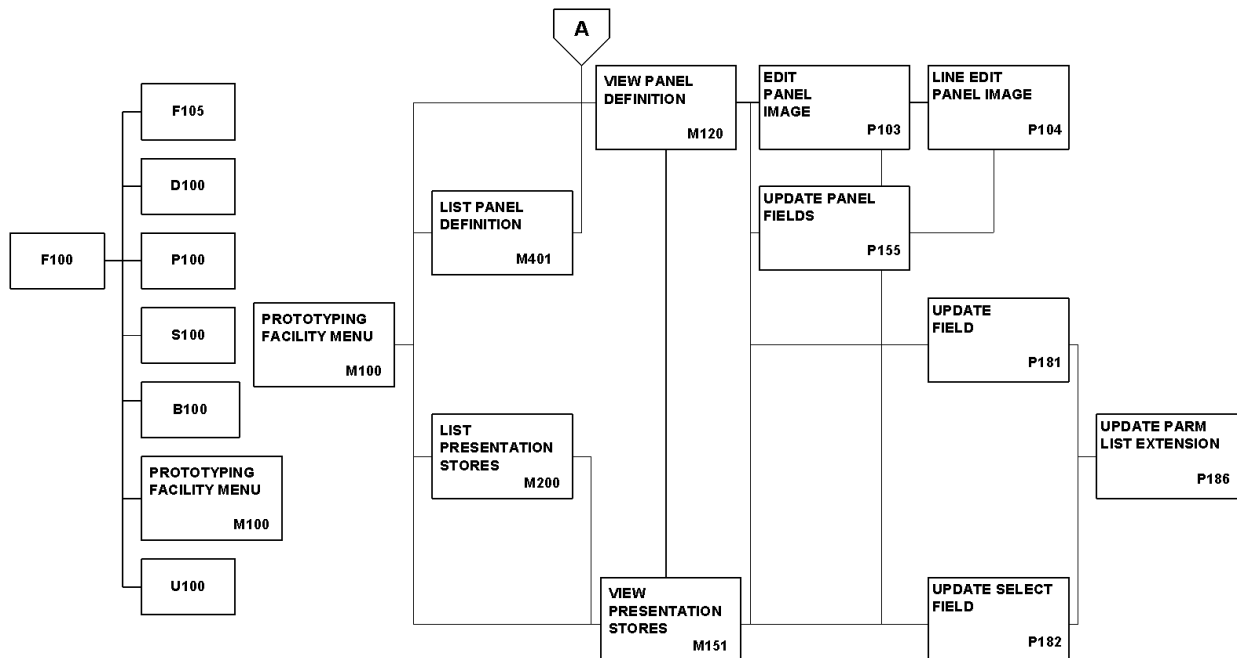
TDF Option 5 Screen Organization (continued)



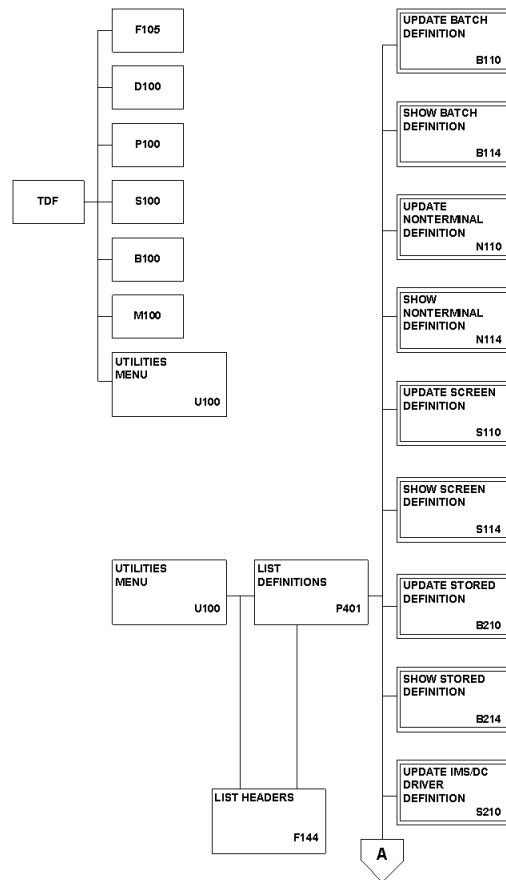
TDF Option 6 Screen Organization

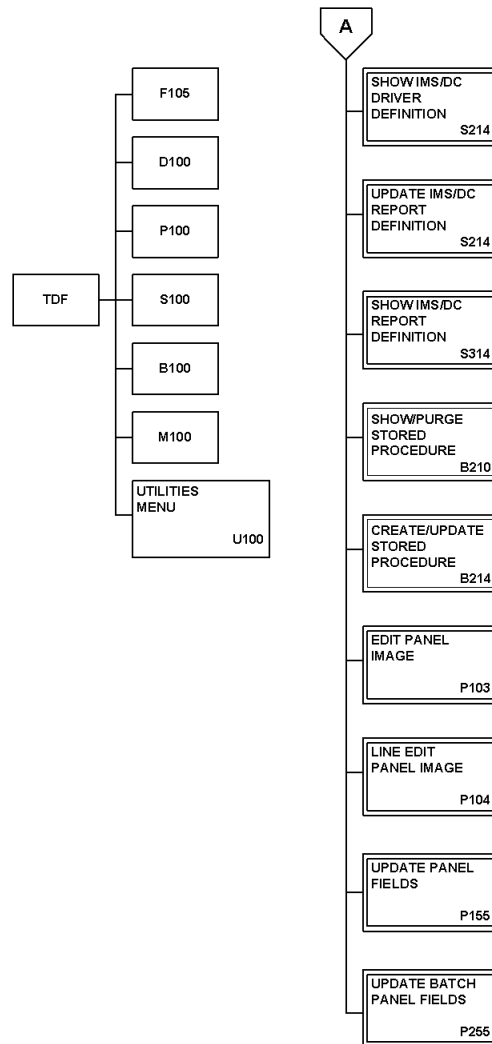


TDF Option 6 Screen Organization (continued)



TDF Option U Screen Organization



TDF Option U Screen Organization (continued)**TDF Main Menu**

You begin the TDF by entering a TSO CLIST, an SPF panel, or a CICS transaction code assigned at installation by your system administrator.

Once you have entered the appropriate request, the TDF first displays several messages relative to data set or data file allocation. You can ignore these messages.

After the last message is displayed, three asterisks (***) appear. At that point, press Enter to get the TDF Main menu shown in the Design Facility Main Menu screen.

Note: From the PWS environment, you access this menu in a different manner.

```

TELON DESIGN FACILITY MAIN MENU *****
COMMAND ==> _____

FUNCTION: __

1 — USER PROFILE MAINTENANCE
2 — DATA ADMINISTRATION
3 — PANEL SPECIFICATION
4 — ONLINE PROGRAM DEFINITION MENU
5 — BATCH PROGRAM DEFINITION
6 — PROTOTYPING FACILITY
U — UTILITIES
X — EXIT

```

Note: This screen is slightly different for the CA Telon/PWS environment.

The TDF Main menu is your entry into all TDF functions. You can request functions by entering the appropriate character:

1. User Profile Maintenance - This allows you to define program defaults (1D), PF keys (1P), and TDF session (1S) defaults.

For further information, see Chapter 4, "User Profile Maintenance."

2. Data Administration - This allows you to create, update, purge, show, and list PSBs, PCBs within a PSB, databases, segments within a database, VSAMs, queues, journals, or sequential files. It also allows read-only access to the DB2 Catalog.

For further information, refer to the *Data Administration* and the *Design Facility Reference* manuals.

3. Panel Specification - This allows you to create, update, purge, show, and list items relative to your panel image and panel definition. This is detailed in Chapters 5 and 6.

4. Online Program Definition - This allows you to access a menu where you can then access either the Online Program Definition menu or the Nonterminal Program Definition menu.

For further information, refer to the *Design Facility Reference*.

The Online Screen Program Definition (4) allows you to create, update, purge, show, and list items relative to your online screen definition, report definition, CICS nonterminal definition and IMS driver definition. This includes defining the IBM environment, database or data file usage, and custom code. This is detailed in Chapter 11.

5. Batch Program Definition - This allows you to create, update, purge, show, and list all items relative to your batch or stored procedure program definition. This is detailed in Chapter 11.
6. Panel images and panel definitions. The prototyping facility is a tool to model panel images and panel definitions. In either case, the purpose is to include the application user in an interactive session for defining an application function and immediately reviewing that function with simple scenarios (sequences of screen samples to simulate an application). The function is used for either prototyping without data mapping or prototyping with data mapping. This is detailed in Chapter 10.
7. Utilities (U) - These allow you to copy, rename, list, or print any panel image, panel definition, screen definition, report definition, driver definition, nonterminal definition, or batch definition. It also allows you to export your screen, report, driver, nonterminal, or batch definition (that is, it translates your TDF created definition into an 80-character CA Telon source code listing). This is detailed in the *Design Facility Reference*.
8. Exit (X) This allows you to exit the TDF and return control to the system.

Parameters

This topic describes the function and usage of the parameters contained on the Main menu.

COMMAND Line

This is the standard CA Telon command line that is located at the top of most TDF screens. You can specify PF key functions and transfer to other screens from this line. The following describes the PF key functions and transferring functions that you can perform from this command line.

Primary Commands

The following table describes the primary commands:

Primary Command	Description
BACKWARD	Pages text BACKWARD one page to show lower line numbers.
CAN	See CANCEL below.

Primary Command	Description
CANCEL	CANCELS data entered on current screen. This functions on screens controlled by submenus. CANCEL returns to the immediately previous screen. You must use CANCEL on each screen until control returns to the submenu and the message CANCEL COMPLETE is displayed.
CAPS	Translates lower case data into upper case.
END	Saves data on the current screen and passes control to the previous screen.
END HOLD	Ends the current TDF session and transfers you to the immediately preceding held session.
FORWARD	Pages text FORWARD one page to show higher line numbers.
HELP	Accesses HELP information for the screen.
HOLD	HOLDS the current screen with its contents and displays the TDF Main menu. You can use the TDF to perform any function at this time. You return to the held screen with END HOLD.
ISPF	Takes you to the ISPF/PDF Main menu, if you executed the TDF CLIST from TSO.
LEFT	Shifts the screen image left to show lower column numbers.
LINE EDIT	Puts the current screen into the LINE EDIT mode. This is only used to update the panel image.
LINE OUT	Exits LINE EDIT mode and returns you to the full screen edit mode.
LOCATE	Displays a screen beginning with the line number or name you specify on the COMMAND line.
MENU	Saves data on current screen and returns control to the TDF Main menu.
NOP	No option.
NULLS	Converts trailing spaces in the member that you are editing. This allows you to enter characters without first having to enter END OF FILE at the end of a line of text where you want to enter characters.

Primary Command	Description
PD	When using the CA Telon image editor, PD protects fields that you defined on the Update Panel Fields screen from updating.
PDF	Takes you to the ISPF/PDF Main menu, if you executed the TDF CLIST from TSO.
PROFILE	Displays information, at the top of the screen, about the current editing session.
RCHANGE	Repeats the last CHANGE command that you entered.
RESET	Cancels all incomplete line commands and '=CHG' indicators.
RESTORE	Cancels all edits that you made since you entered the current editing session or since you last entered the SAVE command, and restores the current editing session to the state it was in when first entered.
RESUME	Ends the current TDF session and transfers you to the immediately preceding held session.
RFIND	Repeats the last find command that you entered.
RIGHT	Shifts the screen image right to show higher column numbers.
SAVE	SAVES data entered; the screen image remains so that you can continue working.
SWAP	Transfers to one of up to three HOLD screens without canceling HOLD with END HOLD.
SWAP EDIT	Transfers you to the Update Panel Fields screen from both panel editor formats; transfers you to the Line Editor format from the Update Panel Fields screen.
TSO	Prefix used to execute a TSO command from TDF (example: TSO TIME).

Transfer Commands

The COMMAND line also allows you to access other menus listed on the TDF Main menu. The following text lists some of the transfer commands that you can enter on the COMMAND line:

- =1 - User Profile Maintenance menu
 - =1D - User Profile Program Defaults
 - =1P - User Profile PF Keys
 - =1S - User Profile Update Session Control
- =2 - Data Administration menu
- =3 - Panel Definition menu
- =4 - Online Program Definition menu
- =5 - Batch Program Definition menu
- =6 - Prototyping Facility menu
- =U - Utilities

In addition, you can use the COMMAND line to access SPF by entering TSO ISPF or TSO PDF (depending upon how you reached the TDF).

Prior to transferring you to another screen, CA Telon performs a SAVE operation, except in the case of the CANCEL command (and, of course, HELP and HOLD). CANCEL (or CAN) prevents CA Telon from performing a SAVE operation when you transfer to another screen.

You can also use the COMMAND line to terminate the TDF session. To terminate the TDF session, simply enter =X.

FUNCTION Parameter

You can specify one of the following functions:

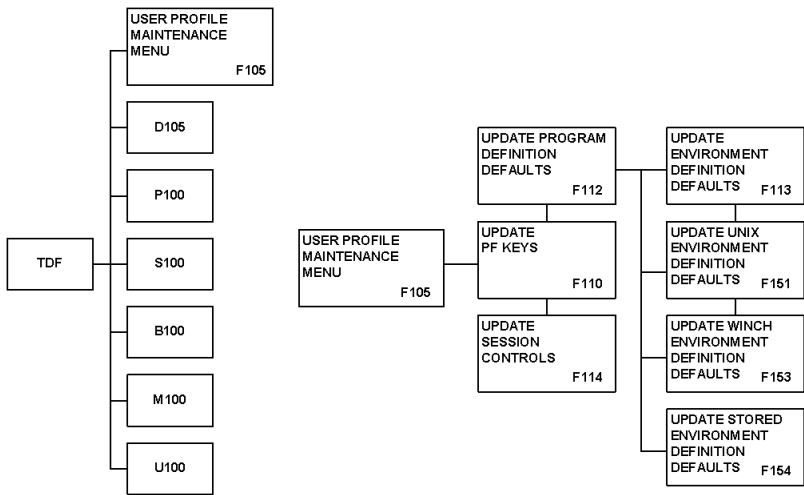
- 1—User Profile Maintenance
 - 1D—User Profile Program Defaults
 - 1P—User Profile PF Keys
 - 1S—User Profile Update Session Controls
- 2—Data Administration
- 3—Panel Specification

- 4—Online Program Definition menu
- 5—Batch Program Definition
- 6—Prototyping Facility
- U—Utilities
- X—Exit

Chapter 4: User Profile Maintenance

This chapter discusses the first option on the TDF Main menu, the User Profile Maintenance option. This option allows you to set default values both for your program and for TDF operations. It also allows you to set or change PF key values.

Organization of the User Profile Maintenance Screens



This chapter introduces you to the User Profile Maintenance menu and user profile maintenance functions. Then it provides an example illustrating how to change default values.

User Profile Maintenance Menu

You reach the User Profile Maintenance menu by entering **1** on the TDF Main menu.

```
USER PROFILE MAINTENANCE MENU *****
COMMAND ==> _____
FUNCTION: D
      D -- DEFINITION DEFAULTS
      P -- PFKEYS
      S -- SESSION CONTROLS
```

The User Profile Maintenance menu allows you to select any of the following functions:

- **D**—Define default values for TDF input fields. These are values such as the transfer work area, custom code name, and the language in which you want CA Telon to create your programs. Entering these values here once allows the TDF to supply them as defaults later.
- **P**—Define the functions the PF keys perform while using the TDF.
- **S**—Define default values valid during this and subsequent TDF sessions. This allows you to define values like export parameters and the panel image size.

The remainder of the chapter takes you through a user profile maintenance example.

Program Definition Defaults Example

This example takes you step-by-step through a session in which you change definition defaults, PF-key defaults, and session controls. For further information on user profile maintenance, refer to the *Design Facility Reference*.

Tasks

This example illustrates these three tasks:

- **Update Program Definition Defaults (screen)**

In this example, you want to change the language default from COBOL to PL/I, and the color and highlight attributes.

- **Update PF-Key Definitions (screen)**

Next, because the PF-key defaults were changed in an earlier session, you will change PF-key definitions back to their original defaults as follows:

- [PF5] and [PF17] ==> RFIND
- [PF7] and [PF19] ==> BACKWARD
- [PF8] and [PF20] ==> FORWARD

- **Update Session Controls (screen)**

Finally, you will change the default generation environment from TSO to CICS and change the SELECT Fields default from | to #.

To change the language default and color and highlight attributes, enter **D** from the User Profile Maintenance menu.

Update Program Definition Defaults

You reach the Update Program Definition Defaults screen by entering **D** on the User Profile Maintenance menu or by entering **1D** on the TDF Main menu. The Update Program Definition Defaults screen allows you to specify default values for various TDF input fields. These are values you would otherwise have to enter on a program-to-program basis.

Note: You can override any default value supplied while you input your program. You do so by simply typing over the supplied values.

```

UPDATE PROGRAM DEFN DEFAULTS *****
COMMAND ==> _____

GENERAL PARAMETERS:
*          LANG   COB   CAPS   ON_
*          OUTIFIL B    HELP  Y    HOLD   Y
*          ALARM  N    EOFKEY Y

CUSTOM CODE DEFINITION MEMBERS:
*          XFERWKA _____
*
          PGMCUST _____

EXTENDED ATTRIBUTES:
*          EATTR N (Y,N)
*          COLOR   HIGHLIGHT   COLOR   HIGHLIGHT
*          EAIN    _____ EAOUT  _____
*          EALIT   _____ EAERR   _____

UPDATE ENVIRON DEFN DEFAULTS _ (C-CICS/I-IMS/T-TSO/B-BATCH/S-STORED)

```

The Update Program Definition Defaults screen shows your installation defaults. To change the language (LANG) to PL/I, type **PLI** over **COB** as shown.

```

UPDATE PROGRAM DEFN DEFAULTS *****
COMMAND ==> _____

GENERAL PARAMETERS:
*          LANG   COB   CAPS   ON_
*          OUTIFIL B    HELP  Y    HOLD   Y
*          ALARM  N    EOFKEY Y

CUSTOM CODE DEFINITION MEMBERS:
*          XFERWKA _____
*
          PGMCUST _____

EXTENDED ATTRIBUTES:
*          EATTR N (Y,N)
*          COLOR   HIGHLIGHT   COLOR   HIGHLIGHT
*          EAIN    RED_____ B_____ EAOUT  GREEN_____ R_____
*          EALIT   BLUE_____ U_____ EAERR   YELLOW_____ B_____

UPDATE ENVIRON DEFN DEFAULTS _ (C-CICS/I-IMS/T-TSO/B-BATCH/S-STORED)

```

You also want to use extended attributes to specify color and highlighting for input, outin, select, output, and literal fields, and fields flagged in error.

To change the extended attributes, you must first type **Y** over the **N** default for the extended attributes field (EATTR), as shown. You can then begin changing the color and highlight attributes.

The EAIN COLOR field allows you to specify the default color attributes for input, outin, and select fields. Optional colors are: BLUE, GREEN, RED, PINK, TURQ (TURQUOISE), YELLOW, and NEUTRAL. If you want these fields to be in red, type **RED** for the **EAIN COLOR** field, as shown. The EAIN HILIGHT field allows you to specify the default highlight attributes for input, outin, and select fields. Options are:

HILIGHT Attribute	Valid Entries	Explanation
BLINK	B, BL, BLINK	Field BLINKs when displayed.
REVERSE	R, RE, REV, REVER, REVRS	Field displays in REVERSE video.
DEFAULT	D, DE, DEF, DEFAU, DEFLT	Field appears in DEFAULT mode.
UNDERLINE	U, UN, UNDER	Field is UNDERLINED.

If you want the fields in which the application user must enter data to blink, type **B** in the EAIN HILIGHT field, as shown.

The EALIT COLOR field allows you to specify the default color attributes for literal fields. The color options are the same as for the EAIN COLOR field. If you want literals to be in blue, type **BLUE** in the EALIT COLOR field, as shown.

The EALIT HILIGHT field allows you to specify the default highlight attributes for literal fields. Options are the same as for the EAIN HILIGHT field.

Because you want the literal field to be underlined, type **U** in the EALIT HILIGHT field shown.

The EAOUT COLOR field allows you to specify the default color attributes for outin and output fields. The color options are the same as for the EAIN COLOR field. If you want outin and output fields to be green, type **GREEN** in the EAOUT COLOR field, as shown.

The EAOUT HILIGHT field allows you to specify the default highlight attributes for outin and output fields. Options are the same as for the EAIN HILIGHT field. If you want output type fields to be in reverse video, type **R** in the EAOUT HILIGHT field, as shown. The EAERR COLOR field allows you to specify the default color attributes for fields flagged in error. The color options are the same as for the EAIN COLOR field. If you want error fields to be in yellow, type **YELLOW** in the EAERR COLOR field as shown.

The EAERR HILIGHT field allows you to specify the default highlight attributes for fields flagged in error. Options are the same as for the EAIN HILIGHT field. If you want error messages to blink, type **B** for this parameter as shown.

When you press Enter or the End PF key, your new entries replace the old. You exit the screen by pressing the End PF key, which records all changes and returns you to either the User Profile Maintenance menu or the TDF Main menu, depending upon how you reached this screen.

Update PF Key Definitions

Next, you want to update PF keys from a previous setting. To do so, you must first access the PF Key Definitions screen.

Entering **P** on the User Profile Maintenance menu or **1P** on the TDF Main Menu returns the Update PF Keys screen.

UPDATE PFKEYS *****

COMMAND ==> _____

PF/PA KEY DEFINITIONS:

PF1 ==> HELP_____

PF2 ==> HOLD_____

PF3 ==> END_____

PF4 ==> END HOLD_

PF5 ==> RFIND_____

PF6 ==> RCHANGE_____

PF7 ==> BACKWARD_

PF8 ==> FORWARD_

PF9 ==> SWAP_____

PF10 ==> LINE EDIT

PF11 ==> SWAP EDIT

PF12 ==> MENU_____

PF13 ==> HELP_____

PF14 ==> HOLD_____

PF15 ==> END_____

PF16 ==> END HOLD_

PF17 ==> CANCEL_____

PF18 ==> NOP_____

PF19 ==> BACKWARD_

PF20 ==> FORWARD_

PF21 ==> SWAP_____

PF22 ==> LINE EDIT

PF23 ==> SWAP EDIT

PF24 ==> MENU_____

PA1 ==> NOP_____

PA2 ==> NOP_____

PA3 ==> NOP_____

CLEAR ==> NOP_____

VALUES: HOLD_____SWAP_____

END_____MENU_____

LEFT_____RIGHT_____

RCHANGE_____

CAPS_____NULLS_____

RESUME_____PDF_____

END HOLD_____HELP_____

LINE EDIT_____

LINE OUT_____

FORWARD_____BACKWARD_____

CANCEL_____SAVE_____NOP

LOCATE_____RFIND_____

PROFILE_____RESET_____RESTORE_____SWAP_____EDIT_____

ISPF_____CAN_____PD_____END ALL

LINE COMMANDS: D I R C M A B O) (X XX F L COLS FS LC G U DG

This TDF screen allows you to define the functions that the PF keys perform while you are using the TDF.

Note: You cannot define the PA keys and the Clear key in TSO. Even if you do so, they will not function.

You can change the PF-key functions at any time. Any changes you make are in effect once you exit the screen. They remain in effect until you change them again.

```

UPDATE PFKEYS *****
COMMAND ==> _____

PF/PA KEY DEFINITIONS:

PF1 ==> HELP_____ PF13 ==> HELP_____ PA1 ==> NOP_____
PF2 ==> HOLD_____ PF14 ==> HOLD_____ PA2 ==> NOP_____
PF3 ==> END_____ PF15 ==> END_____ PA3 ==> NOP_____
PF4 ==> END HOLD_ PF16 ==> END HOLD_ CLEAR ==> NOP_____
PF5 ==> RFIND_____ PF17 ==> CANCEL_____
PF6 ==> RCHANGE_____ PF18 ==> NOP_____
PF7 ==> BACKWARD_____ PF19 ==> BACKWARD_____
PF8 ==> FORWARD_____ PF20 ==> FORWARD_____
PF9 ==> SWAP_____ PF21 ==> SWAP_____
PF10 ==> LINE EDIT PF22 ==> LINE EDIT
PF11 ==> SWAP EDIT PF23 ==> SWAP EDIT
PF12 ==> MENU_____ PF24 ==> MENU_____

VALUES: HOLD SWAP END HOLD HELP FORWARD BACKWARD
        END MENU LINE EDIT CANCEL SAVE NOP
        LEFT RIGHT LINE OUT LOCATE RFIND
        RCHANGE
        CAPS NULLS PROFILE RESET RESTORE SWAP EDIT
        RESUME PDF ISPF CAN PD END ALL

LINE COMMANDS: D I R C M A B O ) ( X XX F L COLS FS LC G U DG
  
```

In this example you want to change PF5 and PF17 to RFIND. To do so, type **RFIND** for the PF5 and PF17 parameters as shown. Next, change PF7/PF19 and PF8/PF20 to **BACKWARD** and **FORWARD** respectively, as shown.

When you press Enter or the End PF key, your new entries replace the old. You exit the PF key Definition screen by pressing the End PF key. This saves all changes and returns you to the User Profile Maintenance menu or the TDF Main menu, depending upon how you entered this screen.

Update Session Controls

TDF Main menu returns the Update Session Controls screen shown below.

```

UPDATE SESSION CONTROLS *****
COMMAND ==> _____

DEFAULT HEADER ==> _____
IMAGE CHARACTERS ==> < (INPUT FIELDS)
                  ==> > (OUTPUT FIELDS)
                  ==> + (OUTIN FIELDS)
                  ==> | (SELECT FIELDS)
                  ==> \ (LITERAL BREAK INDICATOR)
FIELD SELECTION  ==> Y (YES/NO - INPUT FIELDS)
                  ==> Y (YES/NO - OUTPUT FIELDS)
                  ==> Y (YES/NO - OUTIN FIELDS)
                  ==> Y (YES/NO - SELECT FIELDS)
                  ==> N (YES/NO - LITERAL FIELDS)
COMPRESS STATEMENTS ==> YES (YES/NO)
ENVIRONMENT         ==> T (C-CICS/I-IMS/T-TSO/B-BATCH
FORMAT              ==> NO_ (YES/NO) PSB ==> NO_ (YES/NO)
DITTO CHARACTER     ==> "
PANEL SIZE          ==> 24 X 080 UPPER CASE LITERALS ==> ON_

USER MODE           ==> 2 (1/2)
DG SEPARATORS       ==> - -
PDSCOPY DSNAMES ==> _____
  
```

This screen allows you to define the default values of various TDF input fields. These input fields are indicator values that the TDF uses to direct internal TDF processing. These fields are not part of the resulting program. To change the default generator environment from **T** (TSO) to CICS, type **C** for the ENVIRONMENT field, as shown.

To change the SELECT field default from | to #, type # for the SELECT FIELDS field, as shown.

```

UPDATE SESSION CONTROLS *****
COMMAND ==> _____

DEFAULT HEADER ==> _____
IMAGE CHARACTERS ==> < (INPUT FIELDS)
                  ==> > (OUTPUT FIELDS)
                  ==> + (OUTIN FIELDS)
                  ==> # (SELECT FIELDS)
                  ==> \ (LITERAL BREAK INDICATOR)
FIELD SELECTION  ==> Y (YES/NO - INPUT FIELDS)
                  ==> Y (YES/NO - OUTPUT FIELDS)
                  ==> Y (YES/NO - OUTIN FIELDS)
                  ==> Y (YES/NO - SELECT FIELDS)
                  ==> N (YES/NO - LITERAL FIELDS)
COMPRESS STATEMENTS ==> YES (YES/NO)
ENVIRONMENT         ==> C (C-CICS/I-IMS/T-TSO/B-BATCH)
FORMAT              ==> NO_ (YES/NO) PSB ==> NO_ (YES/NO)
DITTO CHARACTER     ==> "
PANEL SIZE          ==> 24 X 080 UPPER CASE LITERALS ==> ON_

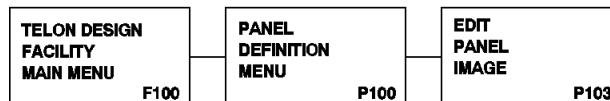
USER MODE           ==> 2 (1/2)
DG SEPARATORS       ==> - -
PDSCOPY DSNAMES ==> _____
  
```

When you press Enter or the End PF key, your new entries replace the old. You exit the screen by pressing the End PF key to save all changes and return you to the User Profile Maintenance menu or the TDF Main menu, depending upon how you reached this screen.

This ends your user profile maintenance example. For additional information, refer to the *Design Facility Reference*.

Chapter 5: Creating a Panel Image

The first task in creating a CA Telon program is to create the panel image. The panel image is the layout of a screen, a report, or a batch event. You create the panel image from the TDF on the Panel Definition menu and the Edit Panel Image screen. The following illustration shows the screen flow that you must follow to create a panel image.



This chapter shows you the steps necessary to create a panel image for both online and batch programs.

TDF Main Menu

To create a panel image, you first need to access the CA Telon Design Facility (TDF). Follow the procedures in effect at your installation to access the TDF. After accessing the TDF, press Enter. CA Telon displays the Main menu.

```
TELON DESIGN FACILITY MAIN MENU *****
COMMAND ==>_____
FUNCTION:      3_
               1 -- USER PROFILE MAINTENANCE
               2 -- DATA ADMINISTRATION
               3 -- PANEL DEFINITION
               4 -- ONLINE PROGRAM DEFINITION
               5 -- BATCH PROGRAM DEFINITION
               6 -- PROTOTYPING FACILITY
               U -- UTILITIES
               X -- EXIT
```

Creating an Online Panel Image

This example shows you how to create a panel image for an Employee Add/Update program. You begin designing the panel image for the Employee Add/Update program by accessing the panel definition menu.

The Panel Definition menu allows you to create and maintain panel images and panel definitions.

To reach the Panel Definition menu:

1. Enter **3** in the FUNCTION field of the TDF Main Menu screen
2. Press Enter.

PANEL DEFINITION MENU *****									
COMMAND ==> _____									
FUNCTION:CR	CR-CREATE	UP-UPDATE	PU-PURGE	SH-SHOW LI-LIST					
ITEM PI	PI-IMAGE	PD-DEFIN	FD-FIELD (UP)		CE-CON SIS (CR,UP)	SL-SEGL OOP (CR,UP,PU)			
MEMBER NAME:									
HEADER	TR _____								
ID	UPDT _____								
DESC	EMPLOYEE ADD/UPDATE SCREEN _____								
ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:									
1. IMAGE	< > + \	(INPUT OUTPUT OUTIN SELECT LIT-BREAK CHARACTERS)							
	24 080	(LINE-COLUMN IMAGE SIZE)							
	U	(UPPER/LOWER CASE LITERALS)							
2. DEFIN	Y Y Y Y N	(INPUT OUTPUT OUTIN SELECT LITERAL FIELDS LISTED)							
3. FIELD	_____	(NAME OR LINE,COLUMN OR "PANEL")							
4. CON SIS	_____	(TYPE - "XFEDIT", "SEGEDIT", OR BLANK FOR LIST)							
	_____	(NAME - IF TYPE SPECIFIED)							
5. SEGL OOP	_____	(TYPE - "FILE" OR "TABLE")							
	_____	(FROM NAME OR LINE,COLUMN)							
	_____	(TO NAME OR LINE,COLUMN)							

To create a panel image:

1. Type **CR** in the FUNCTION field.
2. Type **PI** in the ITEM field.

The three fields under MEMBER NAME apply to the screen (panel) you are creating. The HEADER and ID fields are used to provide labels to specifically identify a screen (in this case, **TR** and **UPDT**—for training add/update).

Note: All CA Telon names are composed of up to six characters. This is divided between a HEADER, which usually indicates a program or application, and an ID, which then indicates the program within the application.

3. Enter the following title for the screen in the DESC field:

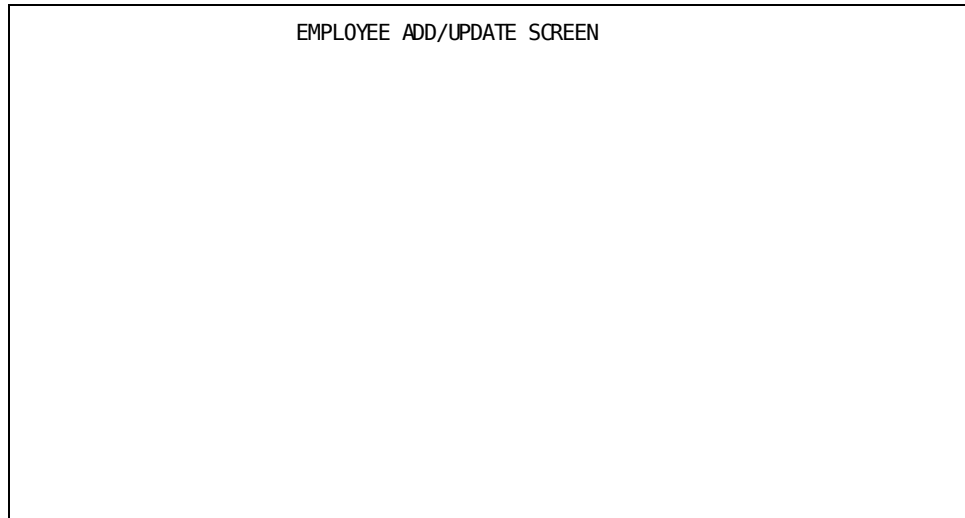
EMPLOYEE ADD/UPDATE SCREEN

Note: Do not include single quotes and ampersands in the DESC field:

The text you enter for DESC automatically appears at the top of the created screen.

4. Press Enter and notice the Edit Panel Image screen appears with **EMPLOYEE ADD/UPDATE SCREEN** displayed at the top, as shown in the Edit Panel Image screen.

You use the Edit Panel Image screen to create or edit your panel image.



You paint an image on the Edit Panel Image screen by using literals, output fields, input fields, output fields, select fields, and the literal break character. The images that you can place on the Edit Panel Image screen are termed *field types*.

Field Types

Anything drawn on the Edit Panel Image screen is automatically defined by CA Telon as one of six types of fields. These fields are defined by their function. They are as follows:

- **Literal**—A literal is any character or series of characters used only to supply information to the application user during program execution. It is not used in processing. The system considers all characters literals except a single quote ('), an ampersand (&), the literal break character described below, or any character used to designate any of the other field types listed below. By default, CA Telon allows blanks between literal groups before designating them separate literals unless you use the literal break character.

Note: The PI Editor and Input will rejoin two literal fields separated by less than three spaces unless these fields have different attributes (i.e. if there is no logical reason for maintaining them as separate fields).

DO NOT use the ampersand (&) or single quote (') on a panel image. If you use either, errors can occur during assembly.

- **Input**—An input field is any character or series of characters keyed by the application user for the first time and processed as input from the screen. In most cases, the field is mapped to a database or work area field. Each input field can be mapped to two separate data fields.
- **Output**—An output field is any character or series of characters that the program displays to the application user. The data for an output field originates from a database, data file, or work field. The application user cannot change the displayed output field.
- **Outin**—An outin field is a combination output and input field. It is processed during program output. The application user can change it. Then it is processed during the input phase.
- **Select**—A select field is a special input field that allows the application user to designate the next screen or processing to be executed. When the application user enters data into a select field, the program branches to a screen or process pre-determined by the programmer. When a screen has more than one select field, the application user must enter data in only one select field. The program displays an error message when data is entered into more than one field.
- **Literal Break**—The literal break character breaks a literal into two literals. Use this character when you want to process parts of a literal or control an attribute. This character is not used often.

Each of these fields is designated by a particular character. These characters, with the exception of the literal break character, cannot be used within literals. The default field designators are as follows:

- Input—<
- Output—>
- Outin—+
- Select—|
- Literal Break—\

To enter field types, specify the length of the field followed by the field type designation (<, >, +, |, \).

You can change the default values. For further information, see Chapter 4, "User Profile Maintenance" for an example of changing TDF defaults.

Once you use a particular set of values for these field designators and save the panel image, you cannot retrace your steps and change the designators. The only way you can change the designators once the panel image is saved is to purge the panel image and start over with new designators.

Design the screen layout by typing in (*painting*) the desired information and data fields. In this example, the screen has 14 data fields.

EMPLOYEE ADD/UPDATE SCREEN			
ID			+6
NAME			+25
DATE OF BIRTH			+8
SEX			+
PHONE			+12
STREET	+25		
CITY	+25		
STATE	++	ZIP	+5
DATE OF EMPLOYMENT			+8
DEPARTMENT			+3
HOURLY RATE			+6
HOURS PER WEEK			+4
>79			

This example is an add/update screen; therefore, outin (+) field types are used as the following discussion illustrates.

If the employee ID already exists on the file, the screen is in *update mode*, and all the fields will be filled with file data. If the employee ID does not exist, the screen is in *add mode*, and the ID field will be filled with the ID received from the previous program, and the DATE OF EMPLOYMENT field will be filled with the current date; all other fields will be empty.

Therefore, all the fields on this screen (except the last) have the potential for receiving data *from* the programs (update function), and sending data *to* the program (add and update functions). The *to* fields are all defined as OUTIN (denoted by the + symbol). The > symbol shows output that CA Telon supplies to the application user (in this case, for error processing).

After pressing [Enter], the completed Employee Add/Update screen displays as shown in the Completed Edit Panel Image screen. Notice how CA Telon has expanded the size of the fields designated in the Initially Specified Panel Image screen.

[illegible]

The specified length for each field appears after each field name. Note the string of > symbols at the bottom of the screen that indicates a message that CA Telon can send to the application user.

Once you finish designing the panel image, press PF3 to save the panel image. This returns you to the Panel Definition menu.

The next CA Telon step is to define the detailed characteristics of the data items that appeared on this screen. This is called *creating a panel definition*, and also is done from the Panel Definition menu. Creating a panel definition is discussed in Chapter 6.

You can prototype the panel image (optional). This is discussed in the chapter "Prototyping Facility."

Creating a Batch Panel Image

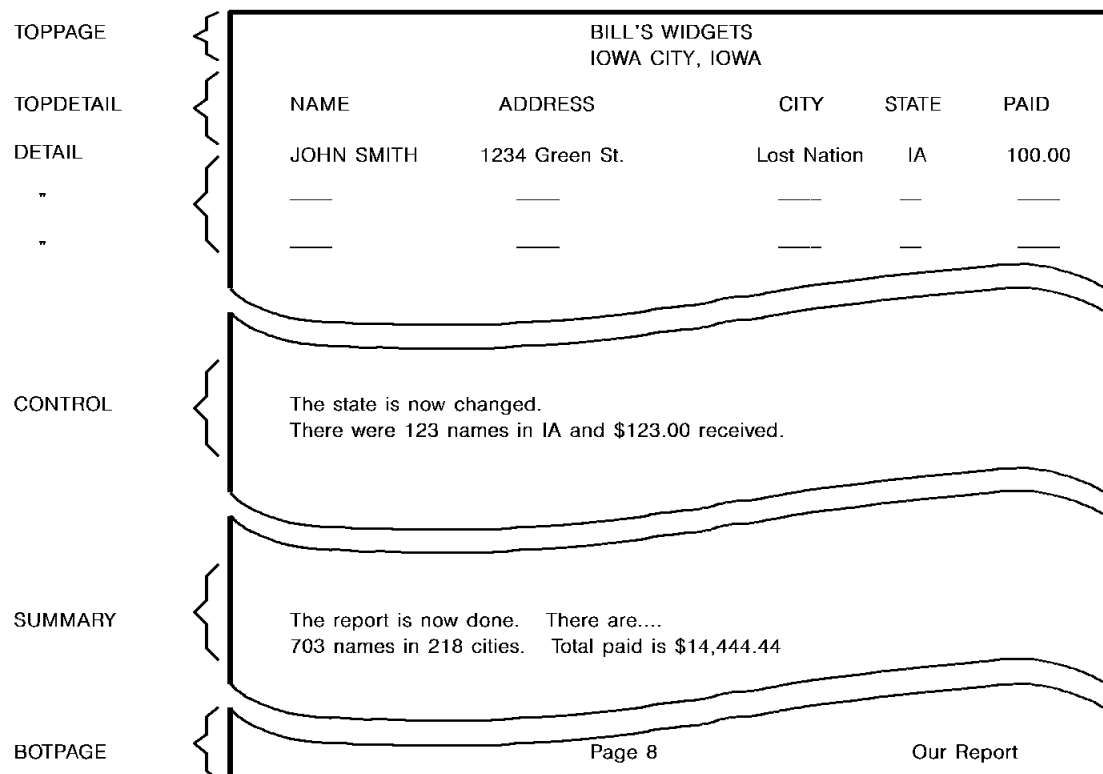
A panel image is required for batch and CICS nonterminal programs that produce a report. Batch and CICS nonterminal programs, such as those that update files or create new files, are not required to produce a report and require only the batch definition. If the program will produce a report, you must create a panel image as well as a panel definition and a batch definition.

The panel image is the most basic unit of your program. It is a layout of a report. You create it by keying in (*painting*) the literal and variable fields on the screen in the format you want the final report to appear. CA Telon uses the panel image to capture field locations, field types, and field lengths. The Batch Reports Events diagram creates a panel image for an employee listing program.

Factors Unique to Batch Panels

When creating a panel image for a batch program, you lay out *groups* of formats. Each group represents a particular event occurring on the report; for example, a TOPPAGE group represents a page heading, and a BOTPAGE group represents the bottom of page detail. CA Telon Batch defines six different events that can occur in your report. Each of the six group types represents one of these events. The following graphic illustrates the six groups.

Batch Report Events



The layout of a report page is not simply painted as it will exist when printed. This is because each page of a report is not necessarily formatted exactly the same as others and because a page usually includes groups of fields whose format is repeated, and should not have to be painted redundantly.

The first step in designing the panel image for the employee report is accessing the Panel Definition menu. The Panel Definition menu allows you to create and maintain panel images and panel definitions.

To reach the Panel Definition menu shown in the following screen.

1. Enter **3** in the FUNCTION field of the TDF Main Menu screen
2. Press Enter

PANEL DEFINITION MENU *****									
COMMAND ==> _____									
FUNCTION:CR		CR-CREATE		UP-UPDATE		PU-PURGE		SH-SHOW LI-LIST	
ITEM	PI	PI-IMAGE		PD-DEFIN		FD-FIELD		CE-CON SIS	
						(UP)		(CR,UP)	
SL-SEGL00P									
(CR,UP,PU)									
MEMBER NAME:									
HEADER		TR_____							
ID		BATC_____							
DESC		EMPLOYEE REPORT_____							
ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:									
1. IMAGE		< > + \		(INPUT OUTPUT OUTIN SELECT LIT-BREAK CHARACTERS)					
		55 133		(LINE-COLUMN IMAGE SIZE)					
		U		(UPPER/LOWER CASE LITERALS)					
2. DEFIN		Y Y Y Y N		(INPUT OUTPUT OUTIN SELECT LITERAL FIELDS LISTED)					
3. FIELD		_____		(NAME OR LINE,COLUMN OR "*PANEL")					
4. CON SIS		_____		(TYPE - "XFEDIT", "SEGEDIT", OR BLANK FOR LIST)					
		_____		(NAME - IF TYPE SPECIFIED)					
5. SEGL00P		_____		(TYPE - "FILE" OR "TABLE")					
		_____		(FROM NAME OR LINE,COLUMN)					
		_____		(TO NAME OR LINE,COLUMN)					

To create a panel image:

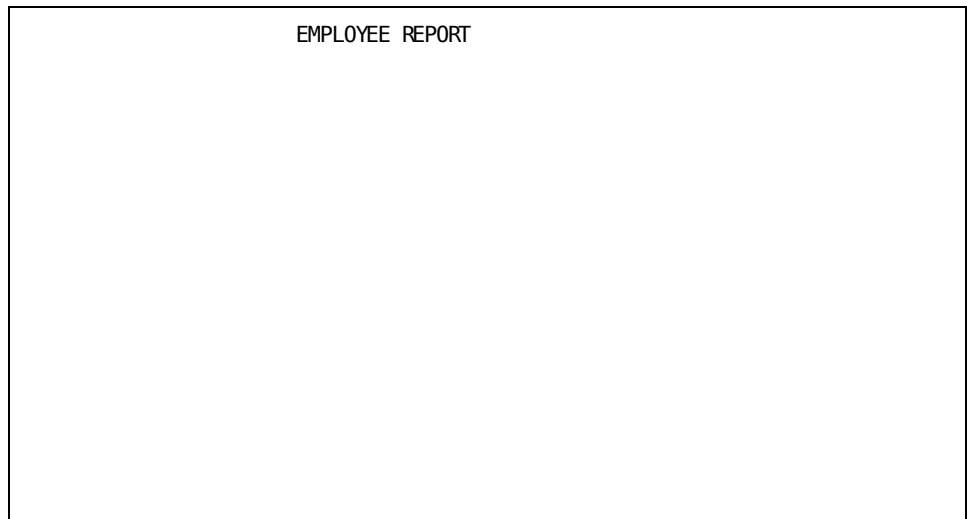
1. Type **CR** in the FUNCTION field
2. Type **PI** in the ITEM field

The three fields under MEMBER NAME apply to the screen (panel) you are creating. The HEADER and ID fields are used to provide labels to specifically identify a screen (in this case, **TR** and **BATC**—for training batch program).

Note: All CA Telon names are composed of up to six characters. This is divided between a **HEADER**, which usually indicates a program or application, and an **ID**, which indicates the program within the application.

3. Specify EMPLOYEE REPORT as the title for the report in the DESC field and press Enter.

Note: The title automatically appears at the top of the created screen as shown in the sample below.



Press PF10 to enter the line edit mode, where it is easier to *paint* the screen. CA Telon then presents you with the Edit Panel Image screen, as shown.

```
LINE EDIT TRBATC.PI * PANEL 055 133 GROUP 001 OF 001 SIZE 000055      COL 002
COMMAND ==>                                SCROLL ==> CSR
0COLS1                                EMPLOYEE REPORT
000002
000003
000004
000005
000006
000007
000008
000009
000010
000011
000012
000013
000014
000015
000016
000017
000018
000019
000020
000021
```

Along with the line numbers, you can display the column numbers.

1. Type **COLS** in the line number area of the first line as shown in the Edit Panel Image screen (line edit mode)
2. Press Enter.

The column numbers appear as shown.

```

LINE EDIT TRBATC.PI * PANEL 055 133 GROUP 001 OF 001 SIZE 000056      COL 002
COMMAND ==>                                SCROLL ==> CSR
000001                                EMPLOYEE REPORT
COLS=>  ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---
000003
000004
000005
000006
000007
000008
000009
000010
000011
000012
000013
000014
000015
000016
000017
000018
000019
000020
000021

```

Field Types

Anything drawn on the Edit Panel Image screen is automatically defined by CA Telon as one of three types of fields. These fields are defined by their function. They are as follows:

- **Literal**—A literal is any character or series of characters used only to supply information to the reader of the report. It is not used in processing. The system considers all characters literals except a single quote ('), an ampersand (&), the literal break character described below, or any character used to designate any of the following fields. By default, CA Telon allows blanks between literal groups before designating them separate literals unless you use the literal break character described below.

DO NOT use the ampersand (&) or single quote (') on a panel image. If you use either, errors can occur during assembly.
- **Output**—An output field is any character or series of characters that the program displays to the application user. The data for an output field originates from a database, data file, or work field. The application user cannot change the displayed output field.
- **Literal Break**—A literal break character breaks a literal into two literals. Use this character when you want to process parts of a literal or control an attribute. This character is not used often.

Each of these fields is designated by a particular character. Output characters cannot be used within literals. The default field designators are as follows:

- Output—>
- Literal Break—\

You can change these default values. For further information, see Chapter 4, "User Profile Maintenance".

The only way you can change the designators once the panel image is saved is to purge the panel image and start over with new designators. Design the report layout by typing (*painting*) the desired information and data fields, that is, the literals and output fields. To enter long output fields quickly and accurately, type a single output field symbol followed by the desired field length. The following screen, a *painted* panel image (using the line edit mode), shows long fields entered in this manner.

```

LINE EDIT TRBATC.PI * PANEL 055 133 GROUP 001 OF 001 SIZE 000055      COL 002
COMMAND ==>                                SCROLL ==> CSR
000001                                EMPLOYEE REPORT
COLS=>  ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---
000003 ID                                NAME, ADDRESS                HRWEEK
000004 >6                                >25                          >5
000005                                >25
000006                                >25
000007
000008
000009
000010                                TOTAL NUMBER OF EMPLOYEES        >3
000011
000012
000013
000014
000015
000016
000017
000018
000019
000020
000021
000022

```


- **SUMMARY**—The SUMMARY group defines the data printed at the end of the report. It is the control break that occurs when there are not more detail lines to print. It can occur only once in a report.
- **BOTPAGE**—The BOTPAGE group defines the data printed at the bottom of a physical page. It is the page footer.

You can use all groups, except SUMMARY, several times in a single CA Telon batch program. You must code at least one DETAIL group in each program. You must indicate the logical groups on the Panel Image screen.

1. Type a **G** on the first line of each logical group as shown in the Report Panel Image screen.
2. Press Enter.

[illegible]

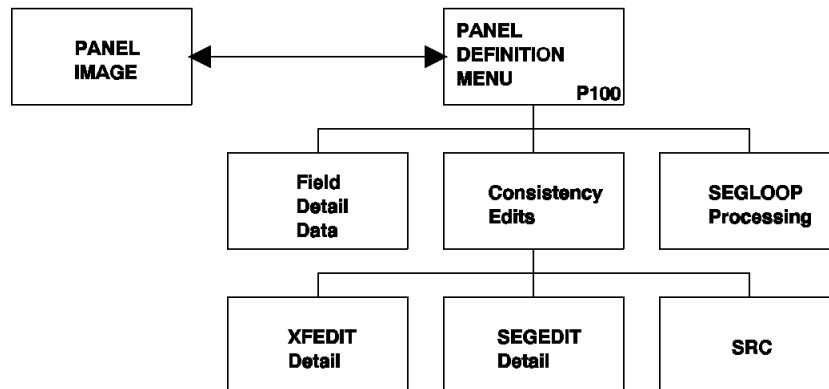
Chapter 6: Creating the Panel Definition

After completing the panel image, your next task is to create a panel definition. The panel definition defines all fields mapped to and from the terminal for a screen definition, or to a printer for a report definition. The panel definition provides data, such as:

- The source of data being mapped to an output field
- The source that the data from an input field will be mapped to
- Whether the application user must enter a particular field
- What edit tests a field must pass
- How CA Telon reformats data during input
- The special attribute characters that you want to use

The panel definition structure shows the portion of your TDF generated program specified in the panel definition. Notice that the panel definition includes the panel image.

Panel Definition Structure



The field characteristics are the first item you must enter into the panel definition. The field characteristics include field mapping, editing, and additional attribute settings.

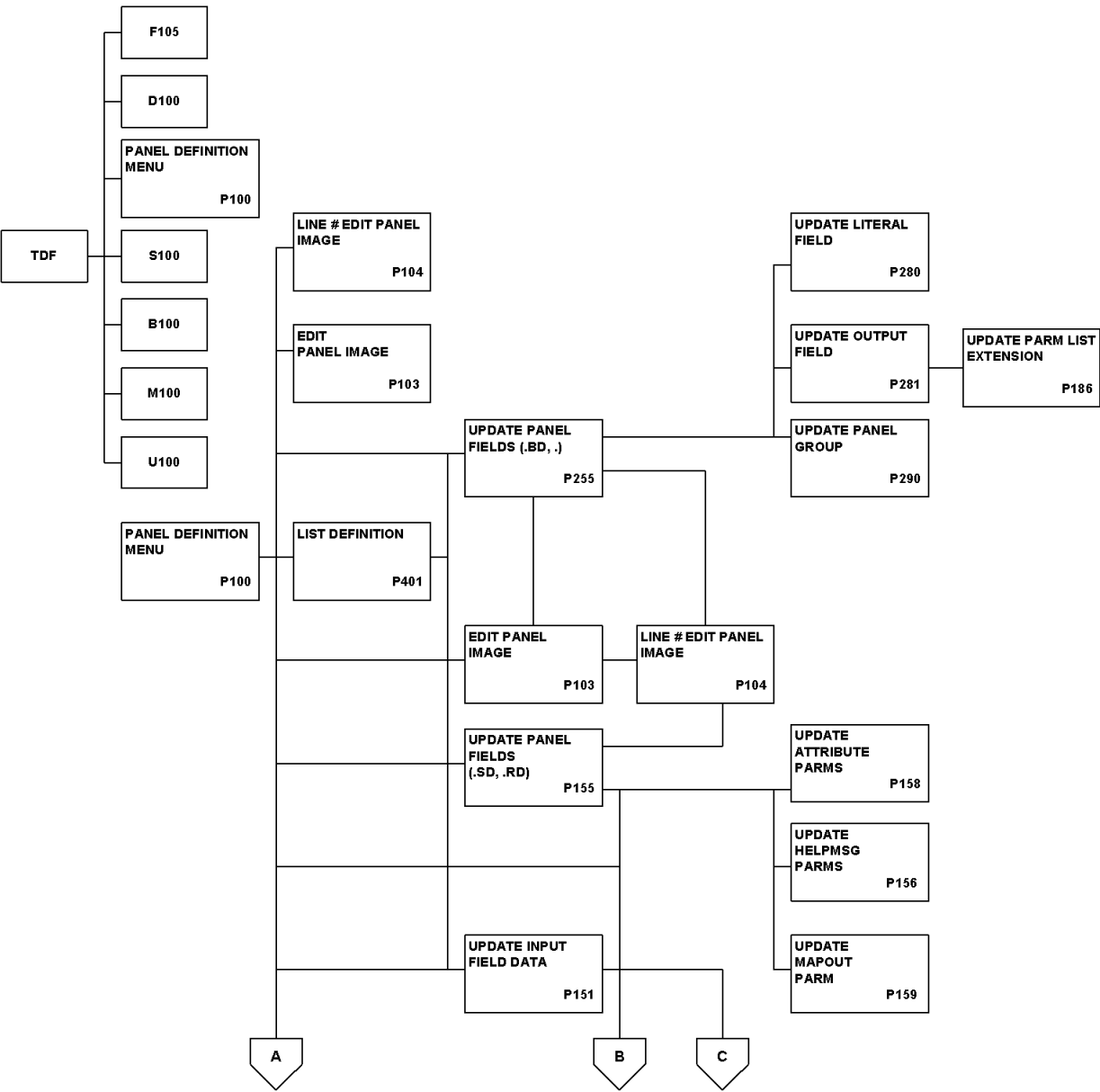
After you specify the field information, specify consistency edits to be included for the panel definition. Consistency edits can be:

- Crossfield validations (the comparison of two or more variable fields)
- Database validations (the checking for the existence or nonexistence of a segment on the database)
- SRC edits (the insertion of COBOL or PL/I statements)

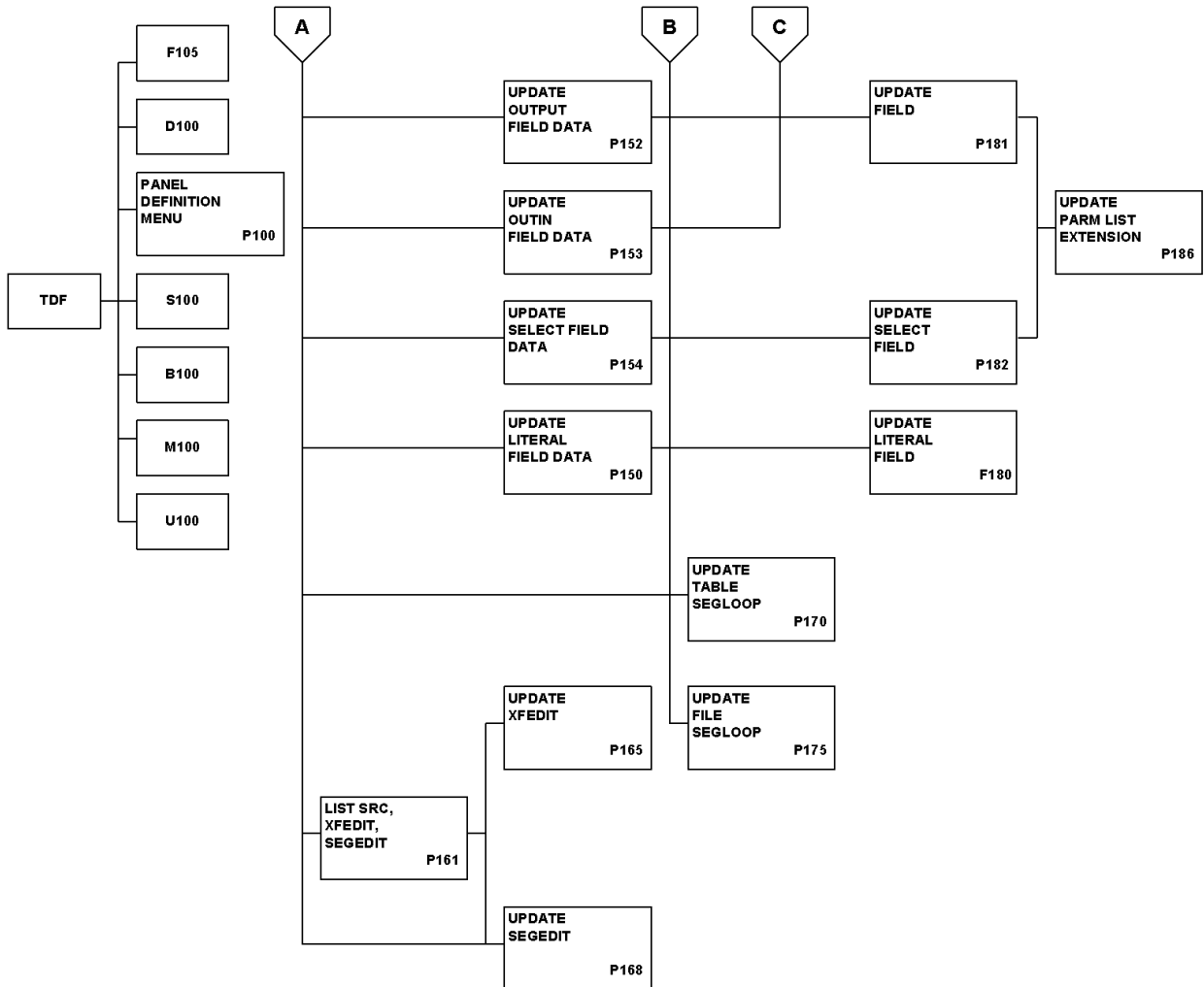
If the panel definition includes the listing of multiple groups of the same type of data, either from the reading of multiple segments or from an array, define a SEGLOOP definition for the panel definition.

To create a panel definition, use the following screen flow. Note that not all programs will use SEGLOOPS and consistency edits. The example contained in this chapter uses consistency edits.

Organization of Panel Definition Screens



Organization of Panel Definition Screens (continued)



Creating an Online Panel Definition

After the panel image is complete, a description of field processing is defined by creating a panel definition. This definition includes internal names for the data defined, logical mapping information, editing requirements for the data, and any special attribute processing required.

To create a panel definition, you must perform the following:

- Specify field input
- Specify further updates
- Define consistency edits

To create a panel definition, enter **CR** (for create) in the FUNCTION field of the Panel Definition Menu screen. This time, however, you must enter **PD** (for Panel Definition) in the ITEM field. The HEADER, ID, and DESC previously entered appears automatically.

Note: You can also transfer directly from a PI (even a newly created one) to a PD by entering the SWAP EDIT command on the PI command line or by pressing a PF key to which the SWAP EDIT command has been assigned (the default is PF11).

```

PANEL DEFINITION MENU *****
COMMAND ==> _____

FUNCTION:CR  CR-CREATE  UP-UPDATE  PU-PURGE      SH-SHOW  LI-LIST

ITEM    PD  PI      PI-IMAGE  PD-DEFIN
                                FD-FIELD  CE-CONSIG  SL-SEGLLOOP
                                (UP)      (CR,UP)    (CR,UP,PU)

MEMBER NAME:
  HEADER      TR____
  ID          UPDT_
  DESC        EMPLOYEE ADD/UPDATE SCREEN_____

ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:
1. IMAGE      < > + | \ (INPUT OUTPUT OUTIN SELECT LIT-BREAK CHARACTERS)
                24 080 (LINE-COLUMN IMAGE SIZE)
                U      (UPPER/LOWER CASE LITERALS)
2. DEFIN      Y Y Y Y N (INPUT OUTPUT OUTIN SELECT LITERAL FIELDS LISTED)
3. FIELD      _____ (NAME OR LINE,COLUMN OR "*PANEL")
4. CONSIG     _____ (TYPE - "XFEDIT", "SEGEDIT", OR BLANK FOR LIST)
                _____ (NAME - IF TYPE SPECIFIED)
5. SEGLLOOP   _____ (TYPE - "FILE" OR "TABLE")
                _____ (FROM NAME OR LINE,COLUMN)
                _____ (TO NAME OR LINE,COLUMN)
  
```

Press Enter and the Update Panel Fields screen appears.

Specify Field Input

You use the Update Panel Fields screen to specify field input.

```

TRUPDT.PD UPDATE PANEL FIELDS *****
COMMAND ==> PAGE 01
OPTIONS ==> ATTRS  HELPMMSG  MAPOUT  _
LINE 001 COL 002 SIZE 024 080
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
0001 EMPLOYEE ADD/UPDATE SCREEN
0002
0003
-----
U  LN  COL  LTH  USE  **NAME**  *FLDTYPE*  *****  DBNAME OR TEXT  *****  REQ MORE
   05  022  006  01
   06  022  025  01
   07  022  008  01
   08  022  001  01
   09  022  012  01
   11  015  025  01
   12  015  025  01
   13  015  002  01
   13  026  005  01
   15  027  008  01
   16  027  003  01
   17  027  006  01
   18  027  004  01
   23  002  079  0U
  
```

The Update Panel Fields screen is divided into two parts. The top part is the panel image window (located between the dashed lines on the screen). You can view any series of three lines from the panel image by changing the number after the LINE field and pressing Enter. To view a particular column through the panel image window, change the number after the COL field, and press Enter.

The bottom part of the Update Panel Fields screen allows you to enter additional information about each field. CA Telon displays the line number (LN), starting column (COL), length (LTH), and usage (USE) of each data field you entered on the panel image. You can then enter the name (**NAME**), field type (**FLDTYPE**), and database name (**DBNAME OR TEXT**) for applicable data fields.

FLDTYPE

Enter a one- to eight-character name for CA Telon to use to map data to or from the DBNAME (below) with or without editing. You enter this when creating or updating the panel definitions.

DBNAME or TEXT

For literals, the literal text appears here as typed by you during *panel painting* (show literals is on). For data, you can enter either the data-name from the copybook or the name of the field you want inserted into working storage.

REQ

Enter a **Y** here to tell CA Telon that the field is required. CA Telon then generates all code to check that the field is always entered. When the field is not entered, CA Telon automatically generates an error message to the application user. If you leave this field blank or enter **N**, CA Telon allows the application user to leave this field blank.

MORE

A plus in this column indicates that you have entered additional information about the data item (on the Update Outin Field screen). Fill out your input, as shown in the Update Fields Panel screen.

```

TRUPDT.PD UPDATE PANEL FIELDS *****
COMMAND ==> PAGE 01
OPTIONS ==> ATTRS  HELPMMSG  MAPOUT  _
LINE 001 COL 002 SIZE 024 080
-----1-----2-----3-----4-----5-----6-----7-----
0001 EMPLOYEE ADD/UPDATE SCREEN
0002
0003
-----
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** REQ MORE
U 05 022 006 OI ID XFER-EMPL-ID
  06 022 025 OI NAME EMPL-NAME Y
  07 022 008 OI DOB DATE EMPL-DOB
      Y
U 08 022 001 OI SEX EMPL-SEX
U 09 022 012 OI PHONE EMPL-PHONE
  11 015 025 OI STREET EMPL-STREET
  12 015 025 OI CITY EMPL-CITY
  13 015 002 OI STATE STATE EMPL-STATE
  13 026 005 OI ZIP EMPL-ZIP
U 15 027 008 OI DOE DATE XFER-TODAYS-DATE Y
  16 027 003 OI DEPT EMPL-DEPARTMENT
  17 027 006 OI RATE DOLLAR EMPL-HOURLY-RATE
U 18 027 004 OI HOURS FLOAT EMPL-HOURS
U 23 002 079 OU ERRMSG1

```

If you want to require the application user to enter data in a particular field, simply enter a **Y** in the REQ column for the field that is required, as shown in the completed Update Panel Fields screen. In this example, you want the application user to enter data for the NAME, DOB, and DOE fields.

Specify Further Updates

To update a particular field further, enter a **U** in the update (U) column corresponding to that field, as shown. When you request multiple updates on this screen, the update screens are presented in order from top to bottom before returning to the Update Panel Fields screen.

In this example, you entered a **U** on the lines corresponding to ID, SEX, PHONE, DOE, HOURS, and ERRMSG1. After all input is complete, press Enter. The first screen accessed is the Update Outin Field screen for the FIELD NAME ID shown in the Update Outin Field screen.

```

TRUPDT.PD UPDATE  OUTIN FIELD *****
COMMAND ==> _____
FIELD NAME ID_____ USAGE OUTIN__ LINE 05 COL 022 LTH 006

GENERAL IN: REQ _ (Y/N/C) HELPMMSG _____
* OUT: PIC _____ OUTATTR _ (Y/N)

MAPPING: DBNAME XFER-EMPL-ID _____
*          OF _____
*          IDBNAME EMPL-ID _____
*          OF _____
*          IDBNAME _____
*          OF _____
*          INIT _____
*          MAPOUT _____

EDIT: FLDTYPE OUT _____ IN _____ PARM LIST EXTENSION _
*          SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*
*
*
*

ATTR:  ATTRPRO Y ATTRINT _____ EACOLOR __ EAHIGH __ EVALIDID __
        FMTEXTIT __ __ FMCNTL=MFS _ (Y/N)
  
```

The above screen assigns the working storage names XFER-EMPL-ID and EMPL-ID to the fields used to map data to and from the ID field on the screen.

In the ATTRPRO field, enter **Y** to specify that the ID field is protected on output.

Now press PF3 to end the update of this field, and CA Telon automatically goes to the Update screen for the next field.

Update SEX Field

The Update Input Field screen is used to update the SEX field. Note that the parameters for this field have been carried over from the Update Panel Fields screen.

```

TRUPDT.PD UPDATE INPUT FIELD *****
COMMAND ==>
FIELD NAME SEX_____ USAGE OUTIN__ LINE 08 COL 022 LTH 001

GENERAL IN: REQ _ (Y/N/C) HELPMMSG _____
* OUT: PIC _____ OUTATTR _ (Y/N)

MAPPING: DBNAME EMPL-SEX_____
*          OF _____
*          DBNAME2 _____
*          OF _____
*          ..... _____
*          OF _____
*          INIT _____
*          MAPOUT _____

EDIT:  FLDTYPE OUT _____ IN _____ PARM LIST EXTENSION __
*      SPEC VALUES_ (FORMAT/CONVERT/VALUES/RANGE)
*      M, F _____
*      _____
*      _____
*      _____

ATTR:  ATTRPRO _ ATTRINT _____ EACOLOR _ EAHIGH _ EVALID _
       FMTEXIT _ _ _ FMTCNTL=MFS _ (Y/N)

```

In the SPEC field of this screen, enter VALUES and then the valid entries for the SEX field—M, F for Male and Female—on the following line. To access the update screen for the PHONE field, press PF3.

Update PHONE Field

With the update screen for the PHONE field displayed, you specify the format used for mapping and editing. Enter the keyword **FORMAT** in the **SPEC** field and **999-XX9-9999** on the line below it, as shown in the Update Input Field screen.

```

TRUPDT.PD UPDATE INPUT FIELD *****
COMMAND ==>
FIELD NAME PHONE__ USAGE OUTIN__ LINE 09 COL 022 LTH 012

GENERAL IN: REQ _ (Y/N/C) HELPMMSG _____
* OUT: PIC _____ OUTATTR _ (Y/N)

MAPPING: DBNAME EMPL-PHONE_____
*          OF _____
*          DBNAME2 _____
*          OF _____
*          ..... _____
*          OF _____
*          INIT _____
*          MAPOUT _____

EDIT:      FLDTYPE OUT _____ IN _____ PARM LIST EXTENSION __
*          SPEC FORMAT (FORMAT/CONVERT/VALUES/RANGE)
*          999-XX9-9999_____
*          _____
*          _____
*          _____

ATTR:      ATTRPRO _ ATTRINT _____ EACOLOR _ EAHIGH _ EAVALID __
           FMTEXIT ____ FMTCNTL=MFS _ (Y/N)
    
```

By specifying the format, you direct the program to check the user-entered phone number for a valid entry of only numbers. CA Telon generates the code to map the positions indicated by an X or 9 into the DBNAME field, and leaves off the special characters. EMPL-PHONE is a 12-byte field consisting of eight numeric, two alphanumeric, and two dash characters. FORMAT also displays this phone number with dashes in the appropriate places (if output).

Press PF3 to save this data, and to continue to the next update screen.

Update DOE Field

The Update OUTIN Field screen for the DOE field update is shown.

```

TRUPDT.PD UPDATE OUTIN FIELD ***** *****
COMMAND ==>
FIELD NAME DOE____ USAGE OUTIN__ LINE 15 COL 027 LTH 008

GENERAL IN: REQ Y (Y/N/C) HELPMMSG _____
* OUT: PIC OUTATTR _ (Y/N)

MAPPING: DBNAME XFER-TODAYS-DATE_____
*      OF _____
*      IDBNAME EMPL-DOE_____
*      OF _____
*      IDBNAME _____
*      OF _____
*      INIT _____
*      MAPOUT _____
EDIT:  FLDTYPE OUT DATE__ IN DATE__ PARM LIST EXTENSION __
*      SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*      _____
*      _____
*      _____
ATTR:  ATTRPRO _ ATTRINT _____ EACOLOR __ EAHIGH __ EVALIDID __
      FMTEXIT ____ FMTQNTL=MFS _ (Y/N)

```

You now specify the input mapping by entering the IDBNAME, EMPL-DOE (EMPL_DOE for PL/I). All other data on the screen was carried over from the Update Panel Fields screen.

Press PF3 to continue to the Update screen for the HOURS field.

Update HOURS Field

On the Update Outin Field screen, you need to specify the acceptable range of number of hours per week that are valid for this field.

```

TRUPDT.PD UPDATE OUTIN FIELD *****
COMMAND ==>
FIELD NAME HOURS__ USAGE OUTIN__ LINE 18 COL 027 LTH 004

GENERAL IN: REQ _ (Y/N/C) HELPMMSG _____
* OUT: PIC OUTATTR _ (Y/N)
_____

MAPPING: DBNAME EMPL-HOURS _____
*          OF _____
*          DBNAME2 _____
*          OF _____
*          ..... _____
*          OF _____
*          INIT _____
*          MAPOUT _____

EDIT:  FLDTYPE OUT FLOAT IN FLOAT__ PARM LIST EXTENSION __
*      SPEC RANGE__ (FORMAT/CONVERT/VALUES/RANGE)
*      20,80 _____
*      _____
*      _____
*      _____

ATTR:  ATTRPRO _ ATTRINT _____ EACOLOR _ EAHIGH _ EAVAILD _
       FMTEXTIT __ __ FMCNTL=MFS _ (Y/N)
  
```

Enter **RANGE** in the SPEC field, and **20, 80** below it to specify a minimum of 20 hours per week and a maximum of 80 hours per week.

The FLDTYPE of FLOAT was entered on the Update Panel Fields screen to allow for a floating point decimal within this range.

Press PF3 to continue to the Update screen for the ERRMSG1 field.

Update ERRMSG1 Field

You can update the ERROR MESSAGE field to highlight error messages when they appear on the screen.

```

TRUPDT.PD UPDATE OUTPUT FIELD *****
COMMAND ==> _____

GENERAL IN: REQ _ (Y/N/C) HELPMMSG _____
* OUT: PIC _____ OUTATTR _ (Y/N)

MAPPING: DBNAME _____
*          OF _____
*          .....
*          OF _____
*          .....
*          OF _____
*          INIT _____
*          MAPOUT _____

EDIT:      FLDTYPE OUT NONE__ IN _____ PARM LIST EXTENSION _
*          SPEC _____ (FORMAT/CONVERT)
*          _____
*          _____
*          _____
*          _____

ATT15      ATTRPRO _ ATTRINT HIGH__ EACOLOR__ EAHIGH__ EVALID__
           FMTEXTIT ____ FMTCTL=MFS _ (Y/N)

```

In the ATTRINT (intensity attribute) field of the Update Output Field screen, enter **HIGH**, to specify that the error message field of the Employee Add/Update Screen should be highlighted. CA Telon places the word **NONE** in the FLDTYPE parameter since **ERRMSG1** is a CA Telon keyword.

Press PF3 to end this update and return to the Update Panel Fields screen. Press PF3 again to return to the Panel Definition menu.

Define Consistency Edits

After painting your panel (CA Telon's panel image) and defining your data (CA Telon's panel definition), you can then easily define cross-field validations and database checks desired for the application.

Cross-field validations compare different fields on a screen against other data on the screen or in the system. If the edit fails, a unique error message is displayed on the screen. Database checks compare key information with the existing database for validation. All of this can be accomplished with manual coding, but would require long hours.

To define consistency edits, you must:

- Specify cross-field edit data
- Specify edit condition

You can perform a cross field edit (XFEDIT) in the Employee Add/Update Screen example to ensure that the new employee is at least 16 years old. To create a consistency edit, you enter **CR** in the FUNCTION field and **CE** in the ITEM field of the Panel Definition Menu screen.

PANEL DEFINITION MENU *****										*****PANEL DEFINITION SAVED*****																				
COMMAND ==> _____																														
FUNCTION:CR					CR-CREATE					UP-UPDATE					PU-PURGE					SH-SHOW LI-LIST										
ITEM		PD		PI		PI-IMAGE					PD-DEFIN					FD-FIELD					CE-CONSIS					SL-SEGL00P				
											(UP)					(CR,UP)					(CR,UP,PU)									
MEMBER NAME:																														
HEADER										TR____																				
ID										UPDT																				
DESC										EMPLOYEE ADD/UPDATE SCREEN_____																				
ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:																														
1. IMAGE		< > + \		(INPUT OUTPUT OUTIN SELECT LIT-BREAK CHARACTERS)																										
		24 080		(LINE-COLUMN IMAGE SIZE)																										
		U		(UPPER/LOWER CASE LITERALS)																										
2. DEFIN		Y Y Y Y N		(INPUT OUTPUT OUTIN SELECT LITERAL FIELDS LISTED)																										
3. FIELD		_____		(NAME OR LINE,COLUMN OR "*PANEL")																										
4. CONSIS		_____		(TYPE - "XFEDIT", "SEGEDIT", OR BLANK FOR LIST)																										
		_____		(NAME - IF TYPE SPECIFIED)																										
5. SEGL00P		_____		(TYPE - "FILE" OR "TABLE")																										
		_____		(FROM NAME OR LINE,COLUMN)																										
		_____		(TO NAME OR LINE,COLUMN)																										

Note: A maximum of 400 lines is allowed on this screen, made up of any combination of SRC, XFEDITS, and SEGEDITs. An SRC line can be a copy statement, which allows the custom code editor to create a member containing more than 400 lines.

List SRC, XFEDIT, SEGEDIT

When you then press Enter, the List SRC, XFEDIT, SEGEDIT screen appears. Enter the following data, as shown below.

In the TYPE field, enter XFEDIT to tell CA Telon to generate a cross field edit.

In the DESCRIPTION field, enter the description **AGECHCK - EMPLOYEE MUST BE AT LEAST 16 YEARS OLD**. AGECHK is the name of your cross-field edit. EMPLOYEE MUST BE AT LEAST 16 YEARS OLD is the description.

```

TRUPDT.PD LIST SRC, XFEDIT, SEGEDIT *****
COMMAND ==> _____ PAGE 01 ****

SEQ TYPE DESCRIPTION (FIRST WORD IS XFEDIT/SEGEDIT NAME) OR STATEMENT CODE
U XFEDIT_ AGECHCK - EMPLOYEE MUST BE AT LEAST 16 YEARS OLD _____
002 _____
003 _____
004 _____
005 _____
006 _____
007 _____
008 _____
009 _____
010 _____
011 _____
012 _____
013 _____
014 _____
015 _____
016 _____

```

To access the Update XFEDIT screen, enter **U** in the SEQ column and press Enter.

Update XFEDIT

You can now specify the edit condition and the error message for the cross-field edit. When the specific DOB input is less than 16 years before DOE (defined as XFER-TODAYS-DATE, the employee is less than 16 years old), the error message **EMPLOYEE MUST BE AT LEAST 16 YEARS OLD** appears in the error message field on the application screen, and the DOB field appears highlighted.

```

TRUPDT.PD UPDATE XFEDIT *****
COMMAND ==> _____

EDIT NAME AGECHK

COPY EDIT BASE: _____ SEGLOOP: ____

EDIT CONDITION: '(EMPL-DOE - EMPL-DOB) < 160000' _____
_____
_____
_____

ERROR MESSAGE: EMPLOYEE MUST BE AT LEAST 16 YEARS OLD _____

HIGHLIGHT FIELDS: DOB, DOE _____

ERRCHAR FIELDS: _____

CURSOR AT FIELD: _____

```

Enter your data as shown in the Update XFEDIT screen.

From these simple parameters, CA Telon generates all the code to check values, to validate data entry, to display the error messages, to highlight the field in error, and to redisplay the screen allowing the user to re-enter the correct data.

When you press PF3, CA Telon returns to the List SRC, XFEDIT, SEGEDIT screen shown. You can add as many cross-field edits as desired. When done, press PF3 again. CA Telon returns you to the Panel Definition Menu screen shown again. When you are done with the edits panel image and Panel Definition Menu screen, enter **=4** in the COMMAND field of this screen to access the Program Definition Menu screen.

PANEL DEFINITION MENU *****

COMMAND ==> 4

FUNCTION:CR

CR-CREATE

UP-UPDATE

PU-PURGE

SH-SHOW

LI-LIST

ITEM

PI

PI-IMAGE

PD-DEFIN

FD-FIELD

CE-CONSIG

SL-SEGL00P

(UP)

(CR,UP)

(CR,UP,PU)

MEMBER NAME:

HEADER

TR

ID

UPDT_

EMPLOYEE

ADD/UPDATE

SCREEN

DESC

ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:

1. IMAGE

< > + | \

(INPUT OUTPUT OUTIN SELECT LIT-BREAK CHARACTERS)

24 080

(LINE-COLUMN IMAGE SIZE)

U

(UPPER/LOWER CASE LITERALS)

2. DEFIN

Y Y Y Y N

(INPUT OUTPUT OUTIN SELECT LITERAL FIELDS LISTED)

3. FIELD

(NAME OR LINE,COLUMN OR "*PANEL")

4. CONSIG

(TYPE - "XFEDIT", "SEGEDIT", OR BLANK FOR LIST)

(NAME - IF TYPE SPECIFIED)

5. SEGL00P

(TYPE - "FILE" OR "TABLE")

(FROM NAME OR LINE,COLUMN)

(TO NAME OR LINE,COLUMN)

You are now ready to define a screen. Creating a screen definition is discussed in Chapter 11.

Creating a Batch Panel Definition

After you design the panel image, you create the panel definition. The panel definition includes all specifications relative to the fields that are mapped to and from the panel image.

A panel definition includes the following:

- Panel image
- Logical group names and types
- Internal names of fields defined
- Source of data mapped to an output field

- Editing requirements for the data such as PIC clauses/field edits
- Any processing characteristics

The source of data mapped to an output field refers to the COBOL or PL/I field that contains the data or calculation to be displayed. It also refers to the database heading that contains the necessary information.

You can begin the panel definition in one of two ways, depending on when you created the panel image.

1. If you created the panel image in an earlier session, enter **3** on the TDF Main menu. To create a panel definition, enter **CR** (for create) in the FUNCTION field of the Panel Definition menu. This time, however, you must enter **PD** (for panel definition) in the ITEM field and the MEMBER NAME (only the HEADER and ID are required, the description is now part of the panel image). This is shown in the Panel Definition Menu screen.
2. If you just completed the panel image and pressed PF11, you are ready to begin the panel definition.

```

PANEL DEFINITION MENU *****
COMMAND ==> _____

FUNCTION:CR  CR-CREATE  UP-UPDATE  PU-PURGE      SH-SHOW  LI-LIST
ITEM  PI      PI-IMAGE  PD-DEFIN
                        FD-FIELD      CE-CONSIG    SL-SEGLLOOP
                        (UP)          (CR,UP)      (CR,UP,PU)
MEMBER NAME:
  HEADER  TR_____
  ID      BATC_____
  DESC    _____

ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:
  1. IMAGE  < > + | \ (INPUT OUTPUT OUTIN SELECT LIT-BREAK CHARACTERS)
                55 133 (LINE-COLUMN IMAGE SIZE)
                U      (UPPER/LOWER CASE LITERALS)
  2. DEFIN  Y Y Y Y N (INPUT OUTPUT OUTIN SELECT LITERAL FIELDS LISTED)
  3. FIELD  _____ (NAME OR LINE,COLUMN OR "*PANEL")
  4. CONSIG _____ (TYPE - "XFEDIT", "SEGEDIT", OR BLANK FOR LIST)
                _____ (NAME - IF TYPE SPECIFIED)
  5. SEGLLOOP _____ (TYPE - "FILE" OR "TABLE")
                _____ (FROM NAME OR LINE,COLUMN)
                _____ (TO NAME OR LINE,COLUMN)
  
```

Specify Logical Groups

After you press Enter, CA Telon displays the Update Panel Fields screen with information from the panel image. The location and lengths of fields entered on the panel image appear here.

The LINE and COL section on the third line of the screen indicates the panel image lines appearing just below in the panel image Display Window (shown in bold text on the screen). Change the LINE and COL values and press Enter to display different lines and columns. The requested line and column number of the panel image will display in the upper-left corner of the window.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
GP
GP
01 002 006 OU
01 018 025 OU
01 054 005 OU
02 018 025 OU
03 018 025 OU
GP
01 044 003 OU

```

On this screen, you will specify:

- Logical group names
- Logical group types
- Field names
- DBnames
- Further updates

At this point, add the logical groups on the Update Panel Fields screen, as shown in the Update Panel Fields - Adding Logical Groups. On the lines that have GP (group) in the USE column, type the name of the group in the NAME column and the group type in the FLDTYPE column. Group types are TOPPAGE, TOPDETAIL, DETAIL, CONTROL, BOTPAGE, and SUMMARY. For more information, see Chapter 5, "Creating a panel image" for further descriptions of these groups.

A group name is important because you can specify multiple groups of the same type (except for SUMMARY).

Adding logical groups

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>                                     PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP TITLE      TOPPAGE
      GP EMPINFO    DETAIL

01 002 006 OU
01 018 025 OU
01 054 005 OU
02 018 025 OU
03 018 025 OU
      GP ENDINFO    SUMMARY

01 044 003 OU

```

Update Logical Groups

Now that you have the group names and the group types, some of the groups might need additional updates. You can use additional updates to specify spacing on the report, such as a number of lines to skip before printing a particular group. Type **U** in the Update column of the EMPINFO line, as shown in the Update Panel Fields (Groups) screen, and press Enter. CA Telon then displays the Update Panel Group screen on which you can specify any other information.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>                                     PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP TITLE      TOPPAGE
      GP EMPINFO    DETAIL

U
01 002 006 OU
01 018 025 OU
01 054 005 OU
02 018 025 OU
03 018 025 OU
      GP ENDINFO    SUMMARY

01 044 003 OU

```

Since the detail group EMPINFO will repeat on the report, you need to enter format information to keep it from running together. In this example you want CA Telon to skip two lines on the report before printing each detail group. Do this by typing **2** in the SKIPBEF field, as shown in the Update Panel Group EMPINFO screen.

```

TRBATC.PD UPDATE PANEL GROUP *****
COMMAND ==> _____

GROUP NAME EMPINFO_ TYPE DETAIL_

GENERAL: SKIPBEF 2__ (NN/PAGE) SKIPAFT ____ (NN/PAGE)
*          FMTCUST _____
*          PRINT _____

DETAIL: TDSKIP ____
*        REPSEQ _____

CONTROL: CTLVAR _____
*        CTLLTH ____ CTLPIC _____
*        MINOR _____

GRP REF: (TOPPAGE, TOPDTL, CONTROL)
*        FORGRP _____
*        _____
*        _____
*        _____

```

Press PF3 to get back to the Update Panel Fields screen. You also need to update the summary group because you want to skip some lines before printing the group to make it stand out on the report. Type **U** in the update field to the left of the summary group and press Enter. CA Telon then displays, as before, the Update Panel Group screen. Type **6** in the SKIPBEF field to skip six lines before printing that group at the end of the report.

```

TRBATC.PD UPDATE PANEL GROUP *****
COMMAND ==> _____

GROUP NAME ENDINFO TYPE SUMMARY

GENERAL: SKIPBEF 6__ (NN/PAGE) SKIPAFT ____ (NN/PAGE)
*          FMTCUST _____
*          PRINT _____

DETAIL: TDSKIP ____
*        REPSEQ _____

CONTROL: CTLVAR _____
*        CTLLTH ____ CTLPIC _____
*        MINOR _____

GRP REF: (TOPPAGE, TOPDTL, CONTROL)
*        FORGRP _____
*        _____
*        _____
*        _____

```

Press PF3 CA Telon returns you to the Update Panel Fields screen which shows the logical groups. Note that + in the MORE column indicates that you specified additional information about that group.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>                                     PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP TITLE TOPPAGE
      GP EMPINFO DETAIL
+
01 002 006 OU
01 018 025 OU
01 054 005 OU
02 018 025 OU
03 018 025 OU
      GP ENDINFO SUMMARY
01 044 003 OU
  
```

Specify DBNames

Now you type the names of the fields and their dbnames in the Update Panel Fields screen. The dbnames are the database access names for the information required in this report. This information is specific to the database you are using.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>                                     PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP TITLE TOPPAGE
      GP EMPINFO DETAIL
01 002 006 OU ID          EMPL-ID
01 018 025 OU NAME        EMPL-NAME
01 054 005 OU HRWEEK      EMPL-HOURS
02 018 025 OU STREET      EMPL-STREET
03 018 025 OU CITY        EMPL-CITY
      GP ENDINFO SUMMARY
01 044 003 OU TOTEMP
  
```

Press Enter. The screen returns as shown.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>                                     PAGE 01
LINE 001 COL 002 ***** SIZE 053 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP      TITLE      TOPPAGE
      +
      GP      EMPINFO DETAIL                                     +
01 002 006 OU ID                                           EMPL-ID
01 018 025 OU NAME                                           EMPL-NAME
01 054 005 OU HRWEEK                                           EMPL-HOURS
02 018 025 OU STREET                                           EMPL-STREET
03 018 025 OU CITY                                           EMPL-CITY
      GP      ENDINFO SUMMARY                                     +
01 046 003 OU TOTEMP

```

Update Output Fields

Some of the fields require further updates. The HRWEEK field requires a PIC clause to display correctly. In the employee database used for this example, the hours per week field contains five characters. The first three characters are for the whole hours up to 999. The next character is the decimal point and the last is the decimal number of hours worked. Type **U** in the Update column of HRWEEK and press Enter, as shown below.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==>                                     PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP      TITLE      TOPPAGE
      GP      EMPINFO DETAIL                                     +
01 002 006 OU ID                                           EMPL-ID
01 018 025 OU NAME                                           EMPL-NAME
U 01 054 005 OU HRWEEK                                           EMPL-HOURS
02 018 025 OU STREET                                           EMPL-STREET
03 018 025 OU CITY                                           EMPL-CITY
      GP      ENDINFO SUMMARY                                     +
01 044 003 OU TOTEMP

```

This brings you to the Update Output Field screen. The FIELD NAME, USAGE, and DBNAME are already displayed. You must add ZZ9.9 to correctly format the output. Press PF3 to save and quit.

```

TRBATC.PD UPDATE OUTPUT FIELD *****
COMMAND ==>

FIELD NAME HRWEEK__ USAGE OUTPUT_ LINE 01 COL 054 LTH 005

MAPPING: DBNAME EMPL-HOURS
*          OF
*          INIT
*          MAPOUT
*          BLANK.WHEN.SAME _ (Y/N)
GENERAL: PIC ZZ9.9

EDIT: FLDTYPE      PARM LIST EXTENSION __ (CR)
*          SPEC      (FORMAT/CONVERT)
*
*
*
*
ALT CNTGRP      SCOPE      GROUP
MAPPING: TOTREF      TOTSIZE      SCOPE      GROUP
*          CALC
*
*
*

```

Press PF3 to return to the previous screen. Notice that + appears in the MORE column after you add the PIC clause to the HRWEEK field.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==> PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP      TITLE      TOPPAGE
      GP      EMPINFO DETAIL
01 002 006 OU      ID      EMPID- ID      +
01 018 025 OU NAME      EMPID- NAME
01 054 005 OU HRWEEK      EMPID- HOURS      +
02 018 025 OU STREET      EMPID- STREET
03 018 025 OU CITY      EMPID- CITY
      GP      ENDINFO SUMMARY
01 044 003 OU TOTEMP

```


The field that will hold the total number of employees is called TOTEMP. TOTEMP will be a counter, so you need to specify additional information about it. Type U in the update column to the left of the TOTEMP field, as shown in the next screen, and press Enter.

```
TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==> PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK
-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP      TITLE  TOPPAGE
      GP      EMPINFO DETAIL
01 002 006 OU ID                      EMPL-ID
01 018 025 OU NAME                      EMPL-NAME
01 054 005 OU HRWEEK                    EMPL-HOURS
02 018 025 OU STREET                    EMPL-STREET
03 018 025 OU CITY                      EMPL-CITY
      GP      ENDINFO SUMMARY
U 01 044 003 OU TOTEMP
```

CA Telon takes you to the Update Output Field screen, where you specify the field name and usage. Enter the name of the group that will be counted in the CNTGRP field. In this case, TOTEMP must count the detail group that was named EMPINFO. TOTEMP also requires a COBOL PIC clause. Use ZZ9 to allow up to 999 with leading zeros suppressed.

```

TRBATC.PD UPDATE OUTPUT FIELD *****
COMMAND ==>

FIELD NAME TOTEMP__ USAGE OUTPUT_ LINE 01 COL 044 LTH 003

MAPPING: DBNAME
*      OF
*      INIT
*      MAPOUT
*      BLANK.WHEN.SAME _ (Y/N)
GENERAL: PIC ZZ9

EDIT: FLDTYPE      PARM LIST EXTENSION __ (CR)
*      SPEC      (FORMAT/CONVERT)
*
*
*
*
ALT CNTGRP EMPINFO__ SCOPE      GROUP
MAPPING: TOTREF      TOTSIZE __ SCOPE      GROUP
*      CALC
*
*
*

```

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==> PAGE 01
LINE 001 COL 002 ***** SIZE 058 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003 ID NAME, ADDRESS HRWEEK

-----
GPTYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP      TITLE      TOPPAGE
      GP      EMPINFO DETAIL
01 002 006 OU ID                      EMPL-ID
01 018 025 OU NAME                      EMPL-NAME
01 054 005 OU HRWEEK                      EMPL-HOURS      +
02 018 025 OU STREET                      EMPL-STREET
03 018 025 OU CITY                      EMPL-CITY
      GP      ENDINFO SUMMARY
01 044 003 OU TOTEMP

```

The information displayed in the Update Panel Fields screen must be correct before you can perform the next step, creating the batch definition. If necessary, correct the placement of fields and groups, or re-enter the Update Panel Fields screen or the Update Panel Group screen to perform further updates.

For more information on the procedures for creating the batch definition, see the chapter "Creating the Program Definition".

Chapter 7: Program Hierarchical Structure

This chapter explains the CA Telon Program Structure using Online Program Hierarchical charts, Batch Program Hierarchical charts, CICS Nonterminal charts, CICS Client Hierarchical charts, and CICS Server Hierarchical charts.

These charts show the logical structure and order of execution of CA Telon programs. They also show you where procedural custom code is executed relative to the generated code.

Chart Conventions

Read the charts top-to-bottom, left-to-right, as you would read a database hierarchical chart. Each box represents either a function, process, or condition, and includes itself as well as all boxes that are below it logically (for example, denoted by a line going out from the bottom). A box with a double line at the left edge denotes that the function is performed by a section (for COBOL) or procedure (for PL/I) named in the box. A box with a double edge around the entire box denotes a place where custom code (also referred to as COPY Code in the charts) can be brought in by a particular CA Telon parameter. The symbols in the upper-right corner determine how a box relates to the box above it, as described below. Although these symbols may seem complex at the outset, they will become clearer as you read and use the charts.

Symbol	Meaning
none	Sequence: Logic is always executed.
o	Selection: One and only one will be done.
*	Iteration: Logic is repeated zero or more times.
?	Optional: Logic may not be generated.
>	Bypassable: Logic is executed only if CONTROL-INDICATOR is space (" "). Control indicator is defined later in this chapter.
Double bar	Identifies a generated COBOL SECTION or PL/I PROCEDURE.

Online Program Hierarchical Charts

This subsection discusses the online program hierarchical charts in terms of a program overview, main processing, output processing, and input processing.

Program Overview

A generated program is logically broken into two main parts: an output part, which reads and formats data to be presented on the screen, and an input part, which processes the terminal user's action (for example, PF key pressed, data entered, and so on). In addition, a program may contain other components related to the teleprocessing environment it runs under (for example, handling the message queue for IMS/DC or the transaction scheduling for CICS). This chapter presents only the output and input parts of the program regardless of the teleprocessing options in effect.

CONTROL-INDICATOR

Before you can fully understand program flow, you must know the function of the CONTROL-INDICATOR. The CONTROL-INDICATOR is a program field that specifies the next action that the program is to perform. In most cases, the CONTROL-INDICATOR is set automatically by CA Telon; however, you can alter the value of the CONTROL-INDICATOR in custom code.

The following table illustrates the CONTROL-INDICATOR settings:

Setting	LIT Data Item	Logic
O	PROCESS-OUTPUT-LIT	PROCESS-OUTPUT
I	PROCESS-INPUT-LIT	PROCESS-INPUT
R	DO-TRANSFER-LIT	DO-TRANSFER
E	DO-WRITE-LIT	DO-WRITE
C	TRANSACTION-COMPLETE-LIT	TRANSACTION- COMPLETE
Z	INITIAL-PROCESS-LIT	INITIAL-PROCESS
space	CONTINUE-PROCESS-LIT	CONTINUE-PROCESS

To alter the program flow using custom code, you must set the CONTROL-INDICATOR to the appropriate value, then branch to the return label of the current section. The following example illustrates how custom code sends an error message to the screen and transfers control to another program.

- To send an error message to the screen: Set the error message and any highlight attributes, MOVE DO-WRITE-LIT to CONTROL-INDICATOR, and GO TO end-of-routine. For example:

```
MOVE 'ERROR-MESSAGE' TO TPO-ERRMSG1
MOVE ERROR-ATTR TO TPO-FIELD-ATTR
MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
GOTO B-100-OUTPUT-INIT-RETURN
```

- To transfer control to another program:

MOVE 'next-program-id' to NEXT-PROGRAM-NAME-ID, MOVE DO-TRANSFER-LIT to CONTROL-INDICATOR, and GO TO end-of-routine. For example:

```
MOVE 'next-program-id' to NEXT-PROGRAM-NAME-ID, MOVE DO-TRANSFER-LIT to
CONTROL-INDICATOR, and GO TO end-of-routine.
```

For example:

```
MOVE 'XXXX' TO NEXT-PROGRAM-NAME-ID
MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
GOTO P-100-PFKEYS-RETURN
```

CONTROL-INDICATOR

Before you can fully understand program flow, you must know the function of the CONTROL-INDICATOR. The CONTROL-INDICATOR is a program field specifying the next action that the program is to perform. In most cases, the CONTROL-INDICATOR is set automatically by CA Telon; however, you can alter the value of the CONTROL-INDICATOR in custom code.

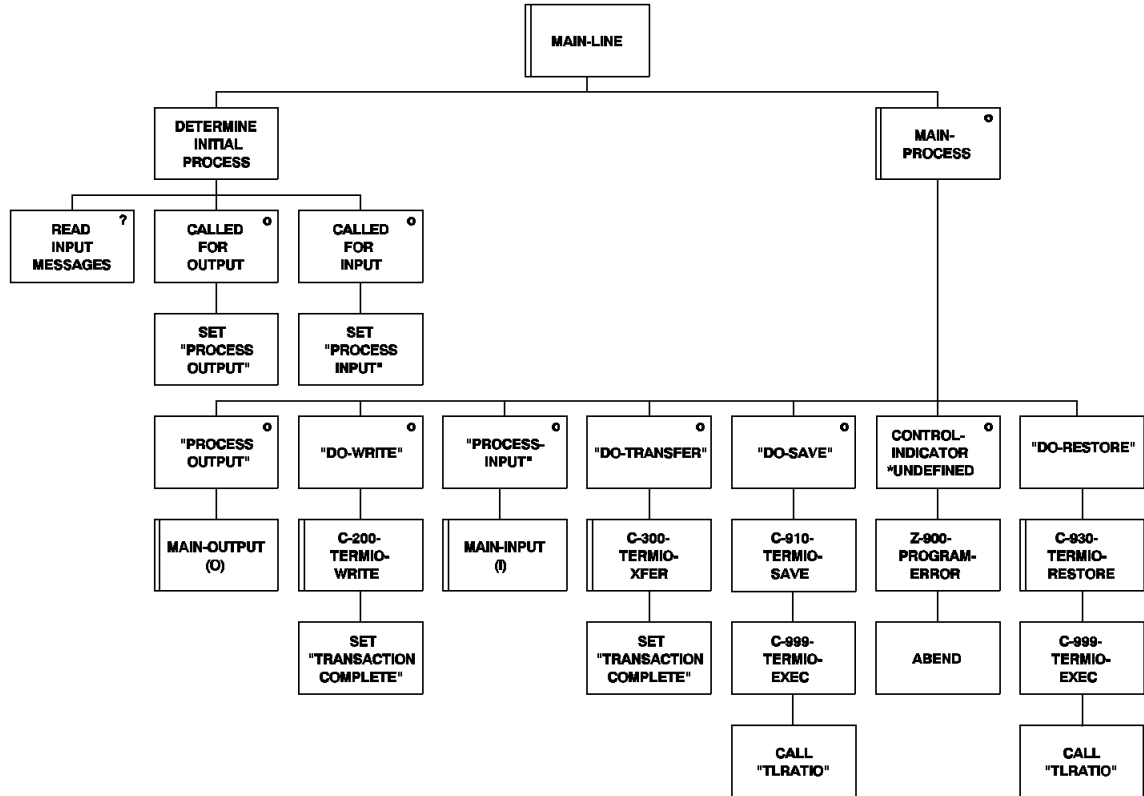
The following illustrates the CONTROL-INDICATOR settings:

Cntrl-Indicator Setting	Literal Fld Action	Value	Action	Default Next Action
GET-TRAN	GET-TRAN-LIT	TRANSACTION	Get the next transaction	Process first detail grpname
PROCESS-DETAIL	PROCESS- grpname1-LIT PROCESS- grpnamen-LIT	grpname1 grpnamen	Format and print a detail line	Process next detail action
END-TRAN	END-TRAN-LIT	TRANSACTEND	Perform termination	Program termination

To alter the program flow using custom code, you must set the CONTROL-INDICATOR to the appropriate value and then branch to the return label of the current section.

Main Processing

The major activities, shown in the hierarchical chart for mainline processing, are: PROCESS-OUTPUT, PROCESS-INPUT, DO-WRITE, and DO-TRANSFER.



MAIN-LINE

In MAIN-LINE processing, CA Telon first performs the DETERMINE INITIAL PROCESS routine to determine whether to set the CONTROL-INDICATOR to PROCESS-INPUT or PROCESS-OUTPUT.

The MAIN-PROCESS routine performs one of the following routines on each iteration depending on the value of the CONTROL-INDICATOR. Once MAIN-PROCESS performs a routine, the value of the CONTROL-INDICATOR is automatically set to specify the next action that the program is to perform. The following table illustrates how the CONTROL-INDICATOR is set:

Set to	Meaning	Next ROUTINE performed
O	PROCESS-OUTPUT	MAIN-OUTPUT to process the output side.
I	PROCESS-INPUT	MAIN-INPUT to process the input side.
R	DO-TRANSFER	C-300-TERMIO-XFER to transfer to the

Set to	Meaning	Next ROUTINE performed
		next screen.
E	DO-WRITE	C-200-TERMIO-WRITE to write a screen.
other values	undefined	ABEND.

As you can see from the above table, the flow of control between PROCESS-OUTPUT, PROCESS-INPUT, DO-TRANSFER, and DO-WRITE is controlled by the CONTROL-INDICATOR field.

The first action you or the application user takes upon designing or entering the program is to determine whether PROCESS-OUTPUT or PROCESS-INPUT is to be performed first. The method of determining which process to perform varies for each environment. Once you determine which process is to be executed, the CONTROL-INDICATOR is automatically set accordingly. The MAIN-PROCESS routine then performs until the CONTROL-INDICATOR is set to TRANSACTION-COMPLETE.

MAIN-PROCESS

Within MAIN-PROCESS, one of the following routines is performed:

- PROCESS-OUTPUT
- DO-WRITE (to perform C-200-TERMIO-WRITE)
- PROCESS-INPUT
- DO-TRANSFER (to perform C-300-TERMIO-XFER)
- CONTROL-INDICATOR *UNDEFINED (to perform an ABEND routine)

As noted earlier, the flow of processing depends on the value of the CONTROL-INDICATOR. For example, when the CONTROL-INDICATOR is set to a value of I (PROCESS-INPUT-LIT), the MAIN-INPUT routine processes the input side of the transaction. The program tests the CONTROL-INDICATOR for a value of space (CONTINUE-PROCESS-LIT) before performing each routine. If all routines are performed and the CONTROL-INDICATOR is still set to space (" "), the CONTROL-INDICATOR is automatically set to R (DO-TRANSFER-LIT). This is the default action after processing input.

DO-TRANSFER performs the C-300-TERMIO-XFER routine to transfer control to the next program (specified by the NEXTPGM parameter in your program definition. The default action after DO-TRANSFER is to SET 'TRANSACTION COMPLETE'. This ends the MAIN-PROCESS routine.

When the CONTROL-INDICATOR is set to **O** (PROCESS-OUTPUT-LIT), the program performs the MAIN-OUTPUT routine. This routine processes the output side of the transaction. The program tests the CONTROL-INDICATOR for a value of space (CONTINUE-PROCESS- LIT) before performing each routine. As on the INPUT side, if the CONTROL-INDICATOR is still set to space (" ") when all the routines have been performed, the CONTROL-INDICATOR is automatically set to E(DO-WRITE-LIT). This is the default action after processing output.

DO-WRITE performs C-200-TERMIO-WRITE to send a message to the terminal. The default action after DO-WRITE is to SET 'TRANSACTION COMPLETE'. This ends the MAIN-PROCESS routine.

Output Processing

The online program hierarchical chart illustrates output processing. In output processing, the MAIN-OUTPUT (O) routine controls output processing for a transaction. Each subsequent routine is performed only if the CONTROL-INDICATOR is space (" "). The default action at the end of MAIN-OUTPUT is SET 'DO-WRITE'. The Output Process may consist of: K-100-HOLD-RESTORE , A-100-OUTPUT-INIT, B-100-OUTPUT-EDITS , and SET 'DO-WRITE' (to end Output Processing).

K-100-HOLD-RESTORE

CA Telon generates K-100-HOLD-RESTORE only if HOLD=**Y** or HELP=**Y** on the Update Screen Parameters screen, or HELPMMSG G is indicated on the Update HELPMMSG Parameter screen, Update Output, Input, Outin Fields screen, or the Update Select Fields screen.

A-100-OUTPUT-INIT

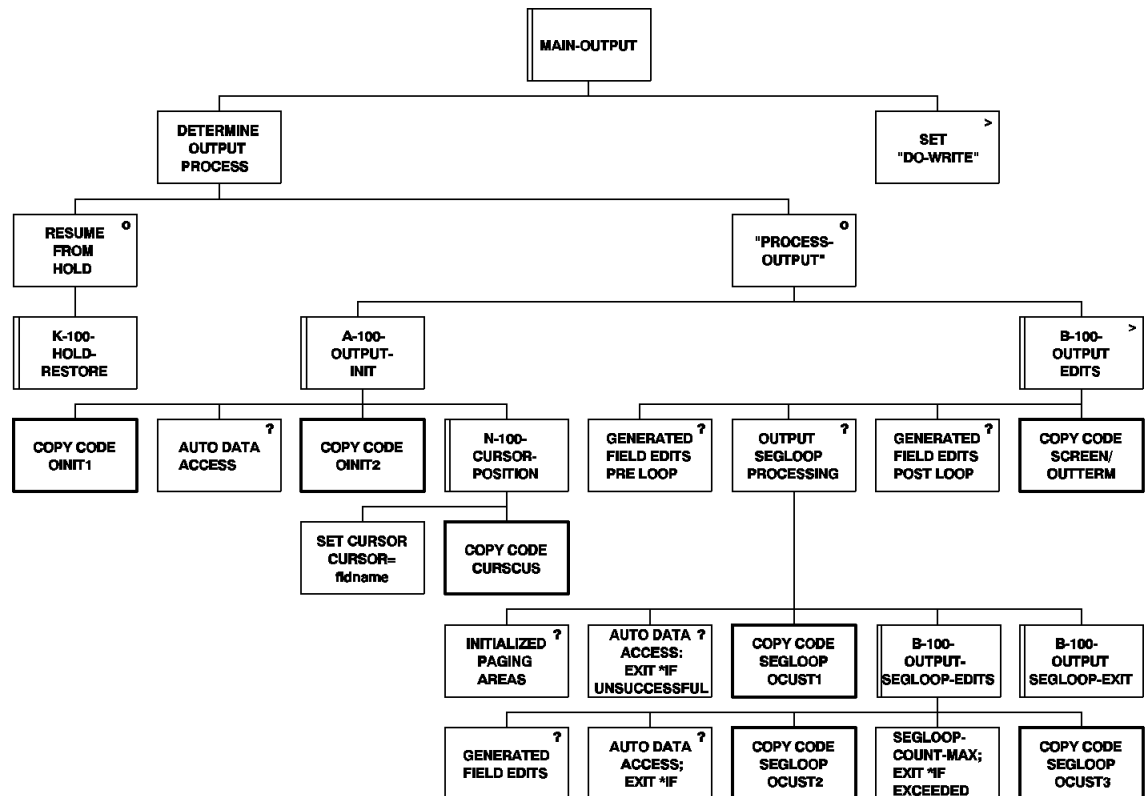
A-100-OUTPUT-INIT comprises any combination of three optional *functions*, and the mandatory cursor-positioning function, as follows: COPY code requested by the OINIT1 parameter, possibly followed by AUTO DATA ACCESS, followed by COPY code requested by the OINIT2 parameter, followed by N-100 CURSOR-POSITION.

The OINIT1 and OINIT2 parameters are specified on the Create/Update Screen Definition screen.

Note: There is no question mark in the upper-right corner of these COPY code boxes. In this case, the use of a question mark would be redundant because, by definition, the code's existence is contingent on whether you include it or not.

The AUTO DATA ACCESS exists if you set up automatic data access on the Create/Update Data Group screen (and for the REQUEST parameter at least one OUTREAD, OIREAD, or UPDATE value is set to **Y**).

N-100-CURSOR-POSITION consists either of COPY Code requested using the CURSCUS parameter on the Create/Update Screen Definition screen, or logic generated to position the cursor to the screen, as defined using the CURSOR parameter on the Create/Update Screen Definition screen. For any single program, you can only specify one of these parameters.



B-100-OUTPUT-EDITS

The B-100-OUTPUT-EDITS section handles editing and formatting of all OUTPUT/OUTIN fields. Additionally, it contains the code generated when an OUTPUT/OUTIN SEGLOOP is required.

The GENERATED FIELD EDITS *function* contains code specified by the FLDTYPE parameter on the Update Panel Fields screen (for example, to edit any OUTPUT or OUTIN fields).

OUTPUT SEGLOOP PROCESSING (SEGLOOP Mapping) exists only if there is an OUTPUT or OUTIN type SEGLOOP specified in the program definition. When SEGLOOP Mapping exists, it can include any of the following: INITIALIZED PAGING AREAS (Page Initialization), AUTO DATA ACCESS, a check for the Success or Omission of the Auto Calls, B-100-OUTPUT-SEGLOOP- EDITS, and B-100-OUTPUT-SEGLOOP-EXIT. These are generated in the order specified. Note that:

- INITIALIZED PAGING AREAS exists only if PAGE=**Y** is specified on the Create/Update File SEGLOOP screen.
- AUTO DATA ACCESS exists if AUTOEXEC BROWSE was specified on the Create/Update Data Group screen. If the automatic access is unsuccessful, the loop is immediately exited to process post-loop generated field edits.
- During execution, B-100-OUTPUT-SEGLOOP-EDITS is repeated zero to n times, where n is the number of groups in the SEGLOOP (as determined by the INCRE parameter on the Create/Update Table SEGLOOP screen). When the repetition is complete, the box following the B-100-OUTPUT-SEGLOOP-EDITS gains control. (Specifically, this is SET 'DO-WRITE'.)

To process each group of fields in the loop, the program executes GENERATED FIELD EDITS through OCUST3. This logic includes the possibility of: GENERATED FIELD EDITS, AUTO DATA ACCESS , and OCUST2 COPY code. A check for maximum SEGLOOP-COUNT is executed next, followed by the possibility of OCUST3 COPY code.

- GENERATED FIELD EDITS exists unless all of the OUTPUT or OUTIN fields specify FLDTYPE=**NONE** on the Update Panel Fields screen, Update Output, Input, Outin Fields screen, or Update Select Field screen. It is possible, but very unusual, that GENERATED FIELD EDITS will not exist.
- AUTO DATA ACCESS exists whenever OCUST1 COPY code exists (for example, when AUTOEXEC is specified on the Create/Update Data Group screen). If the auto access is unsuccessful, the loop is immediately exited to process post-loop generated field edits.
- SEGLOOP-COUNT-MAX checks to determine if the information retrieved from AUTO DATA ACCESS fits on the current screen. If not, AUTO DATA ACCESS is terminated; otherwise, if the loop continues back to GENERATED FIELD EDITS, OCUST3 COPY code receives control first.

Edits are performed for any fields defined after the loop.

- B-100-OUTPUT-SEGLOOP=EXIT exits from OUTPUT SEGLOOP PROCESSING.

OUTTERM, if specified on the Create/Update Screen Definition, is processed next.

SET 'DO-WRITE'

Finally, SET 'DO-WRITE' gains control and writes the screen to the terminal, thereby completing MAIN-OUTPUT.

At this point, output processing is complete and input processing of the screen can begin.

Output Processing Cross Reference

The following table summarizes output processing.

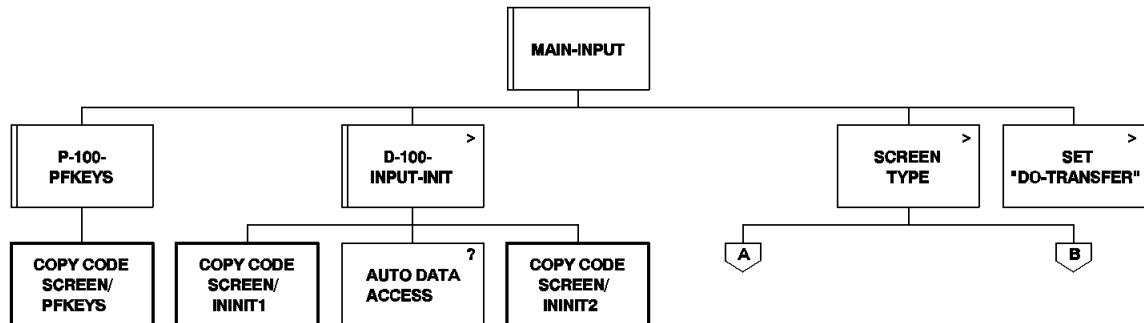
Process	Screen	Parm
C-300-TERMIO-XFER	Create/Update Screen Definition	NEXTPGM
K-100-HOLD-RESTORE	Update Screen Parameters	HOLD, HELP
	Update HELPMMSG Parameter	HELPMSG
	Update Output, Input, Outin	HELPMSG
	Update Select Fields	HELPMSG
OINIT1	Create/Update Screen Definition	OINIT1
OINIT2	Create/Update Screen Definition	OINIT2
AUTO DATA ACCESS	Create/Update Data Group	LABEL, REQUEST
CURSCUS	Create/Update Screen Definition	CURSCUS
CURSOR	Create/Update Screen Definition	CURSOR
GENERATED FIELD EDITS	Update Panel Fields Screen	FLDTYPE
	Update Output, Input, Outin	FLDTYPE
	Update Select Fields	FLDTYPE
OUTPUT SEGLOOP PRO-CESSING	Panel Definition	if SEGLOOP was created
INITIALIZED PAGING AREAS	Create/Update File SEGLOOP	PAGE
AUTO DATA ACCESS	Create/Update Data Group screen	AUTOEXEC BROWSE
B-100-OUTPUT-SEGLOOP-EDITS	Create/Update Table SEGLOOP	INCRE
OCUST1	Create/Update Data Group	LABEL, REQUEST
	Create/Update Table SEGLOOP	OCUST1

Process	Screen	Parm
	Create/Update File SEGLOOP	OCUST1
OCUST2	Create/Update Data Group	LABEL, REQUEST
	Create/Update Table SEGLOOP	OCUST2
	Create/Update File SEGLOOP	OCUST2
OCUST3	Create/Update Data Group	LABEL, REQUEST
	Create/Update Table SEGLOOP	OCUST3
	Create/Update File SEGLOOP	OCUST3
OUTTERM	Create/Update Screen Definition	OUTTERM

Input Processing

The input process, as shown in hierarchical chart for input processing, consists of P-100-PFKEYS, D-100-INPUT-INIT, a bypassable Screen Type check, and a set DO TRANSFER.

Note: If an error is detected during input processing, an error message is sent to the application user's terminal.



P-100-PFKEYS

P-100-PFKEYS comprises the possibility of PF-key processing inherent to the screen (such as paging PF keys for list processing). If you define a SEGLOOP, CA Telon automatically creates this section in your program.

P-100-PFKEYS is followed by the COPY code requested by the PFKEYS parameter on the Create/Update Screen Definition screen.

D-100-INPUT-INIT

D-100-INPUT-INIT consists of the following:

- Possible Custom COPY Code requested by the ININIT1 parameter on the Create/Update Screen Definition screen
- AUTO DATA ACCESS requested using the Create/Update Data Group screen
- Custom COPY Code requested by the ININIT2 parameter on the Create/Update Screen Definition screen

D-100-INPUT-INIT is bypassable depending on processing performed in P-100-PFKEYS.

Screen Type

The path to be traversed after SCREEN TYPE is determined here. If there are any select fields on the screen, the path includes that shown under CONTAINS SELECT FIELDS; beginning with J-100-SELECT.

If there are no select fields, the path begins at CONTAINS NO SELECT FIELDS and includes E-100-INPUT-EDITS, X-100- CON SIS-EDITS, and H-100-INPUT-TERM.

SCREEN TYPE is bypassable depending on processing performed in P-100-PFKEYS.

CONTAINS SELECT FIELDS

CONTAINS SELECT FIELDS consists of J-100-SELECT, followed by PERFORM M-100 FOR FIELD HELP, DETERMINE OPTION SELECTED, and J-100-SELECT OPT- fldname.

J-100-SELECT contains the following components:

- PERFORM M-100 FOR FIELD HELP , which exists if HELP=**Y** on the Update Screen Parameters screen.
- DETERMINE OPTION SELECTED

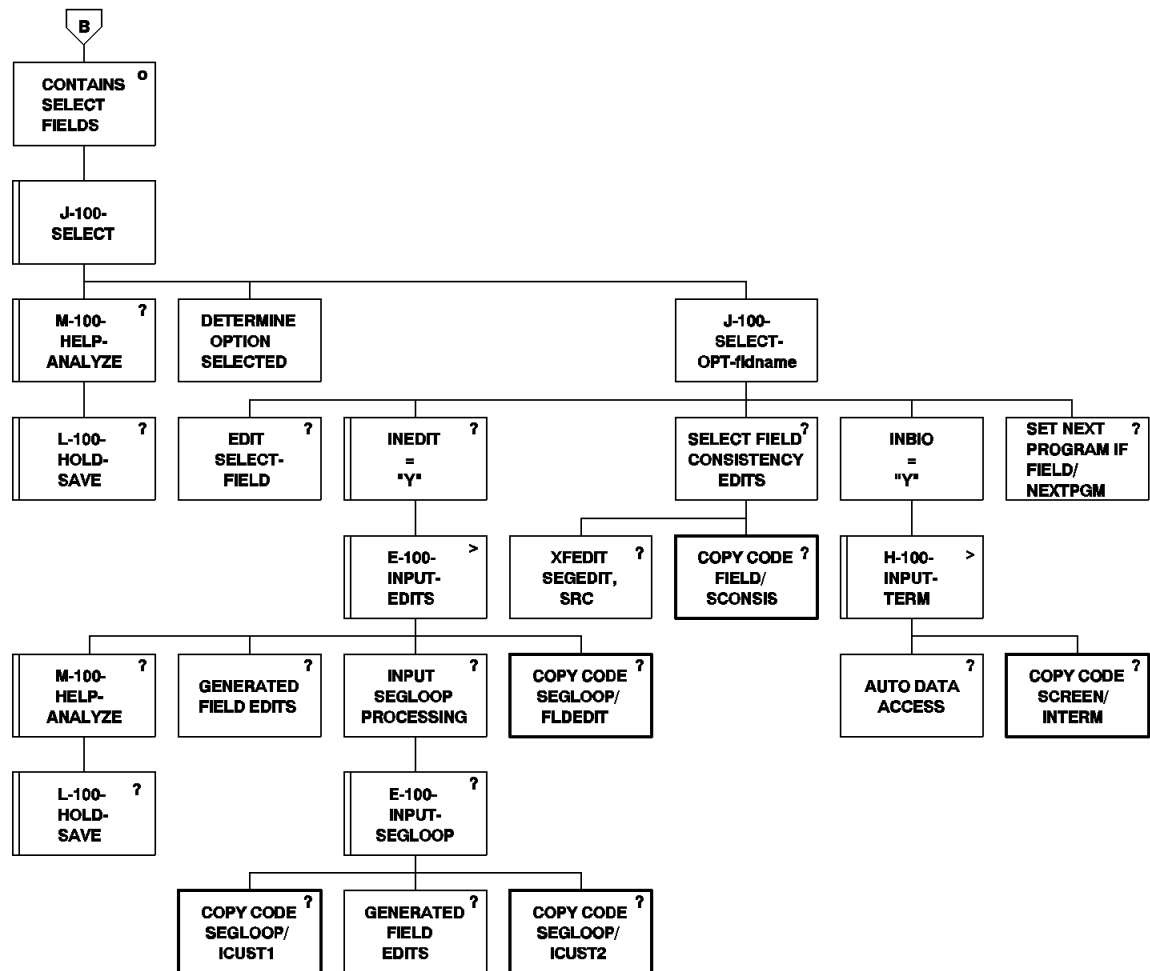
- J-100-SELECT-OPT-fldname , which exists if a non-blank value was entered for the SELECT FIELDS parameter on the Create/Update Screen Definition screen.

Following J-100-SELECT-OPT-fldname are EDIT-SELECT-FIELD, PERFORM E-100 generated FIELD/INEDIT logic, SELECT FIELD CONSISTENCY EDITS , PERFORM H-100 INDBIO logic, and SET NEXT PROGRAM NEXTPGM logic. Note that all of this logic is conditional.

- EDIT-SELECT-FIELD exists based on specifications entered on the Panel Definition menu.
- The E-100 custom code member specified by the FLDEDIT parameter on the Create/Update Screen Definition screen performs at this point if INEDIT=Y was specified on either the Update Select Field screen or the Update Select Parameters screen.
- SELECT FIELD CONSISTENCY EDITS exists only if one of the following conditions is true: a XFEDIT, SEGEDIT, or SRC non-procedural statement is defined on the Panel Definition menu, or if the SCONSIS parameter is specified on either the Update Select Field screen or the Update Select Parameters screen.
- The H-100 FIELD/INDBIO logic exists only if INDBIO=Y was specified on either the Update Select Field screen. If INDBIO=Y, the H-100 logic specified for the INTERM parameter on the Create/Update Screen Definition screen performs.

SET NEXT PROGRAM NEXTPGM logic exists only if the NEXTPGM parameter was specified on the Update Select Field screen.

Note: NEXTPGM is ignored if SELECT fields exist.



CONTAINS NO SELECT FIELDS

CONTAINS NO SELECT FIELDS consists of the following possibilities:

E-100-INPUT-EDITS, X-100-CONSIS-EDITS, and H-100-INPUT-TERM.

E-100-INPUT-EDITS consists of M-100-HELP-ANALYZE, GENERATED FIELD EDITS, INPUT SEGLOOP PROCESSING (SEGLOOP mapping), and COPY code based on the FLDEDIT parameter.

- M-100-HELP-ANALYZE exists only if HELP=Y on the Update Screen Parameters screen, and you code the HELPMMSG parameter on either the Update HELPMMSG Parameter screen, Update Output, Input, Outin Fields screen, or the Update Select Fields screen for at least one INPUT or OUTIN field for this screen (thereby invoking the HELP facility for the field).

L-100-HOLD-SAVE exists only if HOLD=Y is specified on the Update Screen Parameters screen. Note that HOLD is required when HELP is being used.

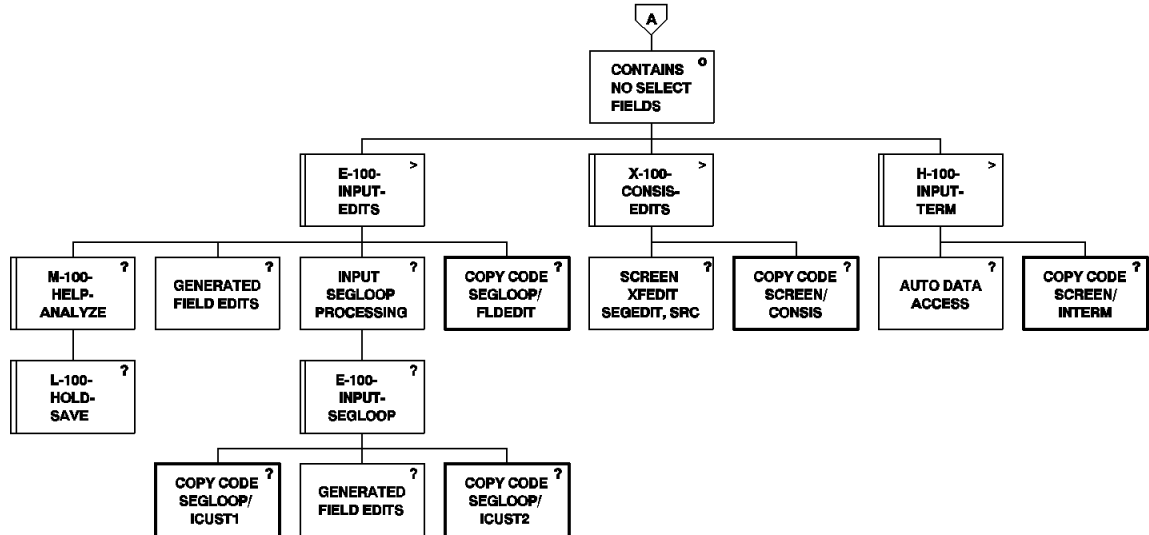
- GENERATED FIELD EDITS exists only if you code the FLDTYPE field on the Update Panel Fields screen for an INPUT or OUTIN panel field.
- INPUT SEGLOOP PROCESSING (SEGLOOP Mapping) exists only if there is an INPUT or OUTIN type SEGLOOP statement in the program definition. INPUT SEGLOOP PROCESSING incorporates E-100-INPUT-SEGLOOP which conditionally processes each input group, performing:
 - The COPY code associated with ICUST1 as specified for the ICUST1 field on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.
 - Any GENERATED FIELD EDITS that were coded using the FLDTYPE field on the Update Panel Fields screen for an INPUT or OUTIN panel field.
 - The COPY code associated with ICUST2 as specified for the ICUST2 field on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.
- The COPY CODE associated with FLDEDIT exists only if you code the FLDEDIT field on the Create/Update Screen Definition screen.
- X-100-CONSIS-EDITS includes consistency edits either generated by the XEDIT, SEGEDIT, or SRC non-procedural statements, or COPY CODE requested by the CONSIS field.

Generated consistency edit code exists only if you specify, on the Panel Definition menu, that a XFEDIT, SEGEDIT, or SRC non-procedural statement has been specified under a non-SELECT type field. The COPY CODE exists only if you specify the CONSIS field on the Create/Update Screen Definition screen.

- H-100-INPUT-TERM incorporates the possibility of AUTO DATA ACCESS specified on the Create/Update Data Group screen and COPY CODE specified for the INTERM field on the Create/Update Screen Definition screen.

SET 'DO-TRANSFER'

SET 'DO-TRANSFER' performs processing to transfer control to the next screen.



Input Processing Cross Reference

The following table summarizes input processing.

Process	Screen	Parm
P-100-PFKEYS	Panel Definition Menu	
PFKEYS	Create/Update Screen Definition	PFKEYS
ININIT1	Create/Update Screen Definition	ININIT1
AUTO DATA ACCESS	Create/Update Data Group	LABEL, REQUEST
ININIT2	Create/Update Screen Definition	ININIT2
PERFORM M-100 FOR FIELD HELP	Update Screen Parameters	SELECT FIELDS
J-100-SELECT OPT-fldname	Create/Update Screen Definition	
EDIT-SELECT-FIELD	Panel Definition Menu	

Process	Screen	Parm
PERFORM E-100	Create/Update Screen Definition	FLDEDIT
	Update Select Field	INEDIT
	Update Select Parameters	INEDIT
SELECT FIELD CON SIS- TENCY EDITS	Panel Definition Menu	
	Update Select Field	SCONSIS
	Update Select Parameters	SCONSIS
PERFORM H-100	Update Select Field	INDBIO
	Update Select Parameters	INDBIO
PERFORM M-100-HELP-ANALYZE	Update Screen Parameters	HELP
	Update HELPMMSG Parameter	HELPMMSG
	Update Output, Input, Outin Fields	HELPMMSG
GENERATED FIELD EDITS	Update Panel Fields	FLDTYPE
INPUT SEGLOOP PRO- CESSING	Program Definition	
ICUST1	Create/Update Table SEGLOOP	ICUST1
	Create/Update File SEGLOOP	ICUST1
GENERATED FIELD EDITS	Update Panel Fields	FLDTYPE
ICUST2	Create/Update Table SEGLOOP	ICUST2
	Create/Update File SEGLOOP	ICUST2
FLDEDIT	Create/Update Screen Definition	FLDEDIT
XFEDIT, SEGEDIT, or SRC	Panel Definition Menu	
CON SIS	Create/Update Screen Definition	CON SIS
AUTO DATA ACCESS	Create/Update Data Group	LABEL, REQUEST
INTERM	Create/Update Screen Definition	INTERM

Batch Program Hierarchical Charts

This subsection discusses the batch program hierarchical charts in terms of:

- Program Overview—The high-level structure of the program.
- Batch Standard Mainline—The processing within the Batch Standard program mainline. This includes a discussion of Section B-1000, B-2000, and B-9000.
- Batch Mainline Sort—The processing within the Batch Mainline Sort program.
- User-Defined Sort—The processing within the Batch User-Defined Sort program.
- Batch Match—The processing within the Batch Match program.
- Batch Merge—The processing within the Batch Merge program.
- Program Control—The TRAN sections to which all CA Telon Batch programs default.

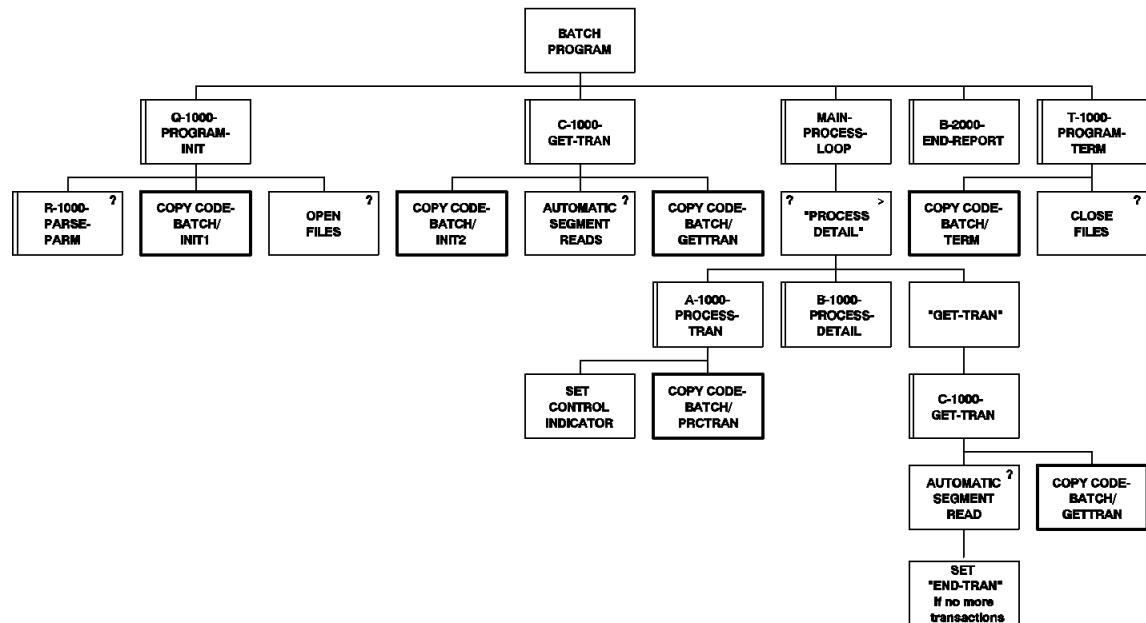
The CA Telon generator produces code for internal sorts, MATCH programs or MERGE programs, depending on entries made in the CA Telon Design Facility. Internal sorts are divided into two categories: mainline and user-defined. Because they are generated differently, this subject presents them separately.

Program Overview

A standard-generated CA Telon batch program consists of three main parts:

1. Input—Program section C-1000
2. Process—Program section A-1000
3. Output—Program section B-1000

The Program Mainline Structure chart illustrates how the sections fit into the batch program. The batch charts and program structures illustrate the report formatting structure subordinate to sections B-1000 and B-2000.



Batch Mainline

The Program Mainline Structure chart shows that the CA Telon Batch program consists of: Q-1000-PROGRAM-INIT, C-1000-GET-TRAN, MAIN-PROCESS-LOOP, B-2000-END-REPORT, and T-1000-PROGRAM-TERM.

Q-1000-PROGRAM-INIT consists of code for parsing JCL PARMS as requested by the PARMS field on the Create/Update Batch Definition screen. This is followed by COPY code, requested by the INIT1 field on the Create/Update Batch Definition screen. This is followed by code for automatic opening of the data sets as requested by the OPEN field on the Update Data Set Record screen. This is followed by any COPY code requested by the INIT2 field on the Create/Update Batch Definition screen.

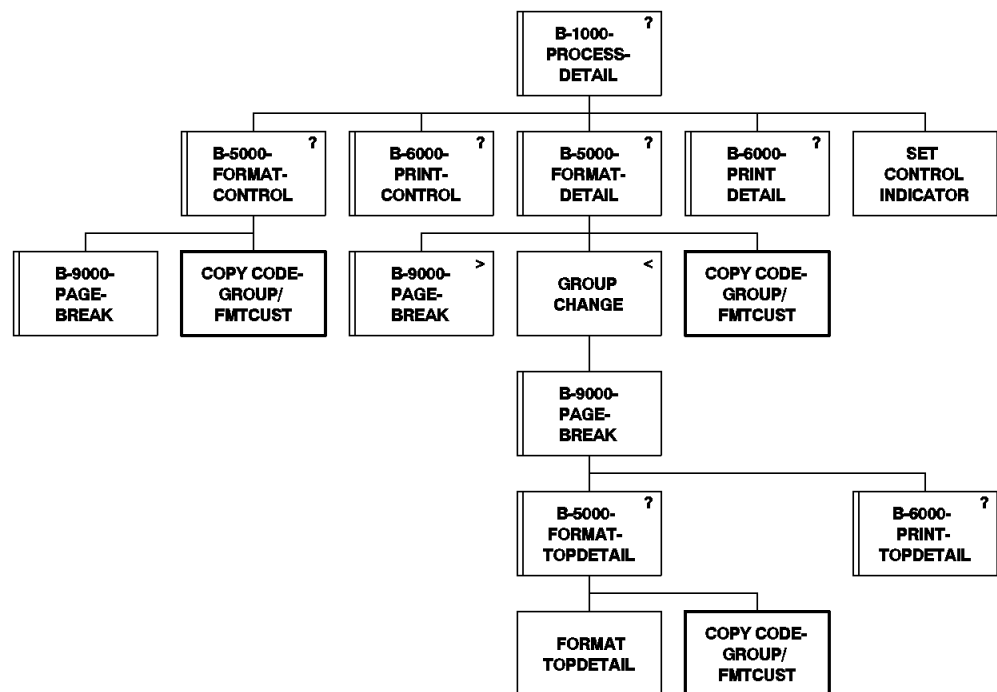
C-1000-GET-TRAN consists of AUTOMATIC SEGMENT READS and COPY code as requested by the GETTRAN field on the Create/Update Batch Definition screen.

MAIN-PROCESS-LOOP consists of: A-1000-PROCESS-TRAN, followed by B-1000-PROCESS-DETAIL (groupname), followed by another C-1000-GET-TRAN.

The A-1000-PROCESS-TRAN contains code which sets the CONTROL-INDICATOR. The CONTROL-INDICATOR controls the program flow (for example, get a transaction, process a transaction and produce detail). The CA Telon default is to read a transaction and write a DETAIL. To override the default group (DETAIL), use custom code to specify the name of the appropriate group. The specified literal (groupname) is moved to the CONTROL-INDICATOR. Following the CONTROL-INDICATOR is any COPY code requested by the PRCTRAN field on the Create/Update Batch Definition screen. See the batch section B-100 chart for more detail on B-1000-PROCESS-DETAIL (groupname).

B-2000-END REPORT - CA Telon generates this section of the program only if the report defines more than one control and/or a summary group. The B-2000-END-REPORT diagram shows more detail.

T-1000-PROGRAM-TERM consists of code for closing the data sets opened in Q-1000-PROGRAM-INIT, followed by any COPY code requested by the TERM field on the Create/Update Batch Definition screen.



B-1000-PROCESS-DETAIL

This routine first checks, formats, and prints any control breaks defined in the report. Secondly, it formats and prints the selected detail based on the CONTROL-INDICATOR. This routine may be performed either in custom code after setting the CONTROL-INDICATOR or by the main process loop after selecting a detail output in A-1000-PROCESS-TRAN. The control is then passed back to MAIN-PROCESS-LOOP to get a new transaction at the conclusion of the processing of this routine.

The batch section B-1000 chart shows that B-1000-PROCESS-DETAIL (groupname) consists of: B-5000-FORMAT-CONTROL (groupname) when you specify a control group, followed by B-6000-PRINT-CONTROL when you specify a control group (for example, this control group is the same control group specified in B-5000-FORMAT-CONTROL), followed by B-5000-FORMAT-DETAIL (groupname), followed by B-6000-PRINT-DETAIL.

Note: At least one Detail Group must be present to generate output (report).

B-5000-FORMAT-CONTROL (groupname) contains a conditional B-9000-PAGE-BREAK. The PAGE-BREAK occurs when you code the SKIPBEF parameter as SKIPBEF=**PAGE** on the Update Panel Group screen, or when CA Telon detects the bottom of the page. This ensures the Control Group(s) prints on the same page with its associated Detail Group(s). B-5000-FORMAT-CONTROL (groupname) occurs once for each Control Group requested. B-5000-FORMAT-CONTROL also contains any COPY code requested by the FMTCUST field on the Update Panel Group screen.

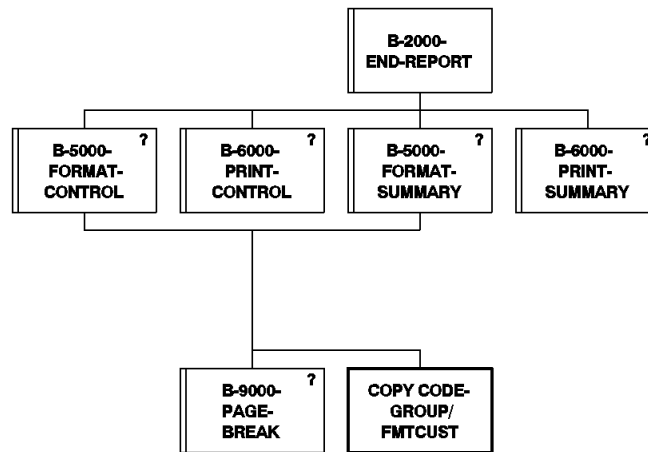
B-6000-PRINT-CONTROL (groupname) contains a conditional B-9000-PAGE-BREAK. The PAGE-BREAK occurs when you code the SKIPAFT field as SKIPAFT=**PAGE** on the Update Panel Group screen. This causes the Control Group associated with the Detail Group to print on the next page. B-6000-PRINT-CONTROL (groupname) occurs once for each Control Group requested. This section immediately follows the corresponding B-5000-FORMAT-CONTROL.

B-5000-FORMAT-DETAIL (groupname) consists of a conditional B-9000-PAGE-BREAK. The PAGE-BREAK occurs when you code the SKIPBEF as SKIPBEF=**PAGE** on the Update Panel Group screen, or when CA Telon detects the bottom of the page. The PAGE-BREAK is followed by a conditional group change. The group change occurs when the group type changes (for example, one detail group to another detail group or any other group type). The PAGE-BREAK is also followed by any COPY code requested by the FMTCUST field on the Update Panel Group screen. The group change, in turn, consists of: a conditional (for example, if there is a group change) B-5000-FORMAT-TOPDETAIL (groupname), followed by B-6000-PRINT-TOPDETAIL. These sections are generated if a Top detail group is defined for this detail group. B-5000-FORMAT-DETAIL occurs once for each detail group requested.

B-5000-FORMAT-TOPDETAIL (groupname) consists of: code to format the top detail group, followed by any COPY code requested by the FMTCUST on the Update Panel Group screen.

B-6000-PRINT-DETAIL (groupname) consists of a conditional B-9000-PAGE-BREAK. The PAGE-BREAK occurs when you code the SKIPAF field as SKIPAF=**PAGE** on the Update Panel Group screen. B-6000-PRINT-DETAIL occurs once for each Detail Group requested and immediately follows the corresponding B-5000-FORMAT-DETAIL.

SET THE CONTROL SWITCH contains code to select another transaction for processing or, if this is the end of processing, control returns to the mainline.



B-2000-END-REPORT

This routine formats and prints any final control break and prints the summary group (if defined) into the report. The batch section B-2000 diagram shows that B-2000-END-REPORT consists of an optional B-5000-FORMAT-CONTROL (groupname). The B-5000-FORMAT-CONTROL section occurs when a control group exists. B-6000-PRINT-CONTROL follows the B-5000-FORMAT-CONTROL, optionally followed by B-5000-FORMAT-SUMMARY (groupname) when a summary group exists, followed by B-6000-PRINT-SUMMARY, followed by B-9000-PAGE-BREAK. B-5000-FORMAT-CONTROL (groupname) contains a conditional B-9000-PAGE-BREAK, which is processed when you code the SKIPBEF parameter as SKIPBEF=**PAGE** on the Update Panel Group screen, or when CA Telon detects the bottom of the page. This is followed by any COPY code requested by the FMTCUST field on the Update Panel Group screen. B-5000-FORMAT-CONTROL occurs once for each control group requested.

B-6000-PRINT-CONTROL (groupname) contains a conditional B-9000-PAGE-BREAK, which is generated when you code the SKIPAF field as SKIPAF=**PAGE** on the Update Panel Group screen. B-6000-PRINT-CONTROL occurs once for each control group requested and immediately follows the corresponding B-5000-FORMAT-CONTROL.

B-5000-FORMAT-SUMMARY (groupname) contains a conditional B-9000-PAGE-BREAK, that CA Telon processes when it detects the bottom of the page.

B-6000-PRINT-SUMMARY (groupname) immediately follows the corresponding B-5000-FORMAT-SUMMARY.

B-9000-PAGE-BREAK is processed when a Botpage and/or Toppage Group exists. The batch section B-9000 contains more detail on B-9000-PAGE-BREAK.

B-9000-PAGE-BREAK

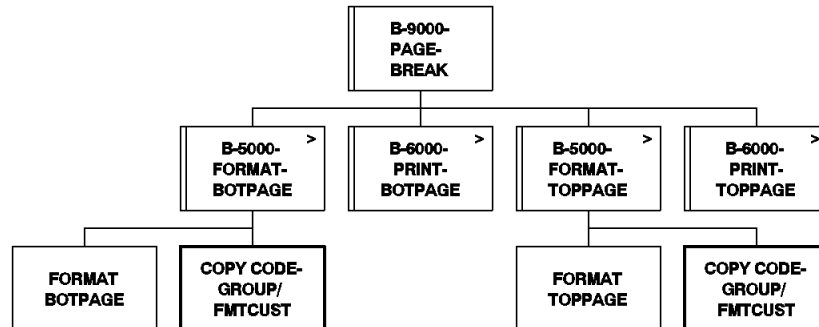
This routine ends the current page and optionally starts a new page in the report. It is performed by detail, control, and summary groups when a new page needs to be started on the report. To start a new page, it performs any toppage and botpage groups required based on the page break type and the page break controlling detail.

The batch section B-9000 diagram shows that B-9000-PAGE-BREAK consists of checks for:

- B-5000-FORMAT-BOTPAGE
- B-6000-PRINT-BOTPAGE
- B-5000-FORMAT-TOPPAGE
- B-6000-PRINT-TOPPAGE

B-5000-FORMAT-BOTPAGE consists of: code to format the bottom of the page, followed by any COPY code requested by the FMTCUST field on the Update Panel Group screen.

B-5000-FORMAT-TOPPAGE consists of: code to format the top of the page , followed by any COPY code requested by the FMTCUST field on the Update Panel Group screen.



The following table summarizes batch processing.

Process	Screen	PARM
B-1000-PAGE-PARM	Create/Update Batch Definition	PARMS
OPEN FILES	Update Data Set Record	OPEN
INIT1	Create/Update Batch Definition	INIT1
INIT2	Create/Update Batch Definition	INIT2
GETTRAN	Create/Update Batch Definition	GETTRAN
PRCTRAN	Create/Update Batch Definition	PRCTRAN
TERM	Create/Update Batch Definition	TERM
B-9000-PAGE-BREAK (all)	Update Panel Group	SKIPBEF
FMTCUST (all)	Update Panel Group	FMTCUST

Mainline Sort

The generator produces all necessary code to execute a basic internal sort in a mainline sort program.

You can construct any of the following four different mainline sort processing structures:

1. PROCEDURE IN/PROCEDURE OUT
2. FILE IN/PROCEDURE OUT
3. PROCEDURE IN/FILE OUT
4. FILE IN/FILE OUT

The mainline sort program, regardless of type, has a new mainline structure with new sections (identified with a *) as well as components of the standard batch program structure:

Section	Use
Q-1000-PROGRAM-INIT	As currently implemented (Presort)
SORT STATEMENT*	Drives sort; optionally invokes MAIN-
SORT-INPUT and MAIN-SORT-OUTPUT	
MAIN-SORT-INPUT*	Input to the sort (as needed)
C-1000-GET-TRAN	Called from within MAIN-SORT-INPUT
MAIN-SORT-OUTPUT*	Output from the sort (as needed)
A-1000-PROCESS-TRAN	Called from within MAIN-SORT-OUTPUT
B-1000-PROCESS-DETAIL	Called from within MAIN-SORT-OUTPUT
B-2000-END-REPORT	As currently implemented (Post sort)
T-1000-PROGRAM-TERM	As currently implemented (Post sort)

As listed above, the four types of mainline sort have either an input procedure or an input file and either an output procedure or an output file. You only need to define input and output files on the TDF. They also must have the same minimum and maximum LRECL as the SORT file.

Input and output procedures, however, allow much more flexibility: you can manipulate data and custom-construct the SORT file. The following describes the structures of the sort input and output procedures:

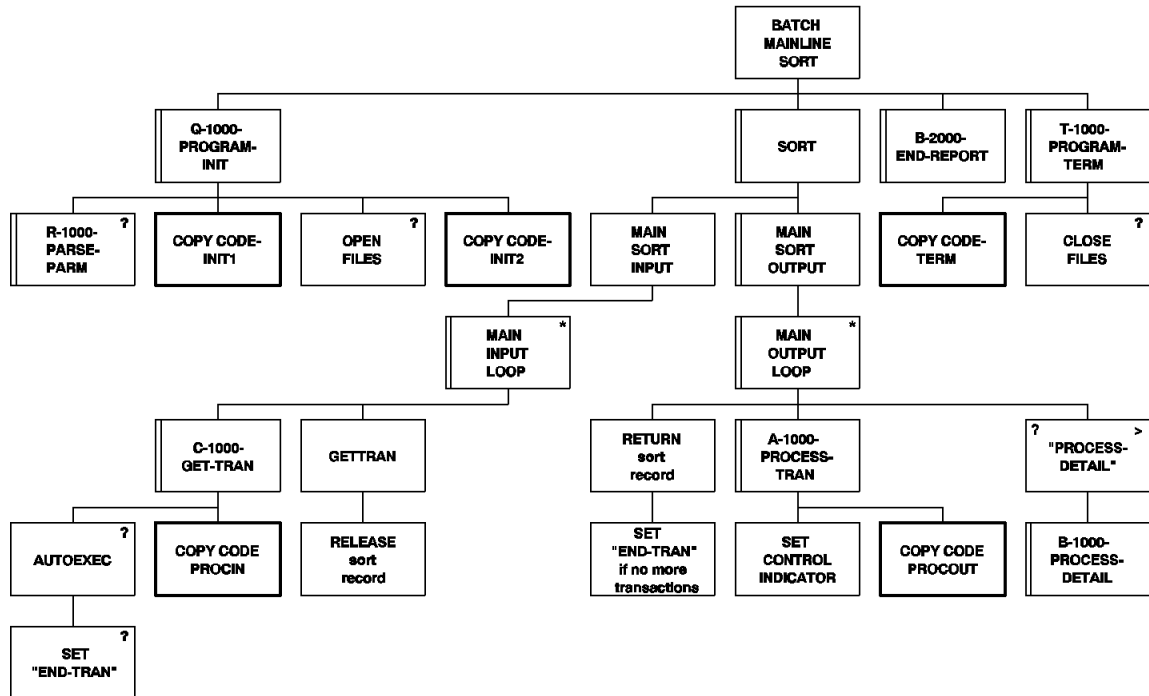
- **PROCEDURE IN:** This procedure is invoked in a loop until the CONTROL-INDICATOR = "END-TRAN". (For PL/I, you must issue a call to PLIRETC with a parameter of 8.) On input to the sort, you can manipulate sort record data in the GETTRAN custom code point, and skip any records you do not want to send to the sort. On return from C-1000, a record is sent to the sort. For COBOL, a record is RELEASEd. For PL/I, a record is RETURNed. PLIRETC with a parameter of 12 is called to signal that there are more records to come; at end-of-file, PLIRETC is called with a parameter of 8.

- **PROCEDURE OUT:** This procedure is invoked in a loop until the CONTROL-INDICATOR = "END-TRAN". (For PL/I, the loop automatically terminates after the return of the last record.)

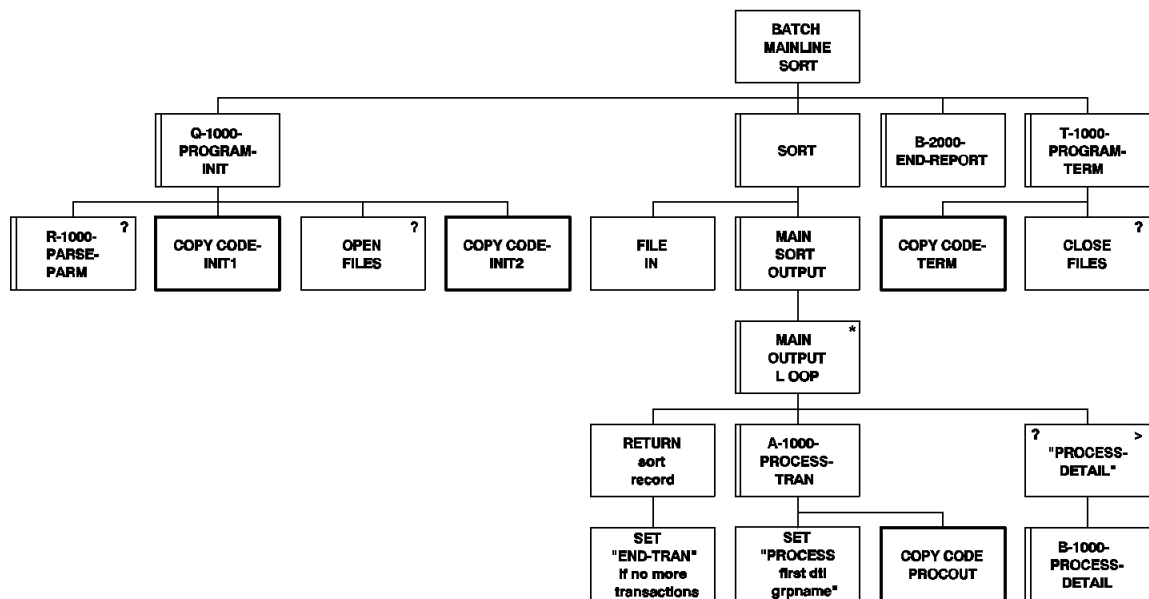
On output, a record is automatically RETURNed from the sort. You can manipulate the output sort record in custom code point PRCTRAN. Then, if the CONTROL-INDICATOR is set to process a detail (PROCESS-groupname-LIT) after the custom code point, B-1000 is called.

In COBOL programs, a CONTROL-INDICATOR value of "END-TRAN" signals the end of execution for the output procedure. For PL/I, PLIRETC with a parameter of 4 must be called at the end of the output procedure to request the return of the next record. Sort output processing is terminated automatically by the sort.

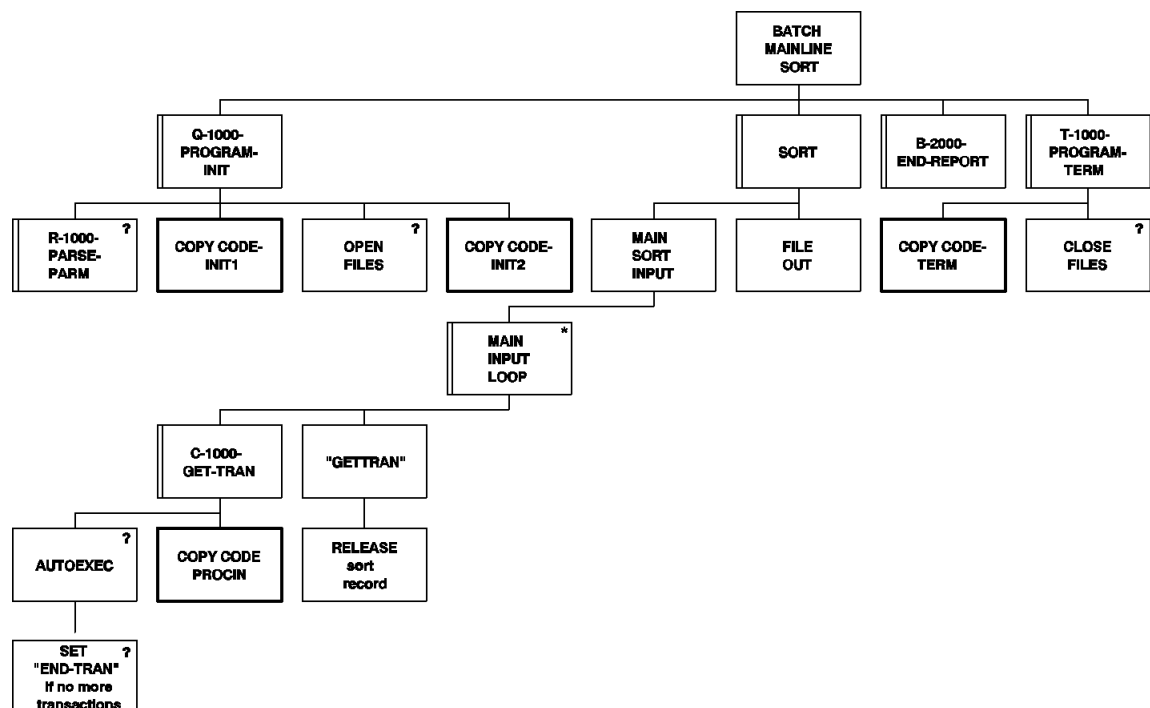
Batch Mainline Sort Program Structure With I/O Proc



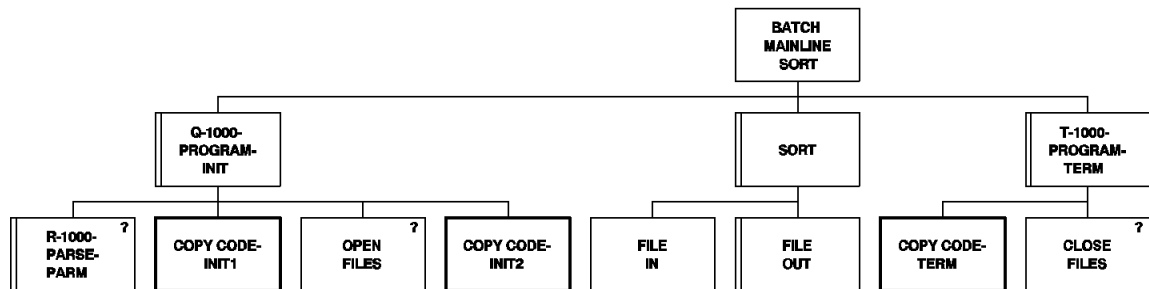
Batch Mainline Sort Program Structure (no input PROC)



Batch Mainline Sort Program Structure (no output PROC)



Batch Mainline Sort Program Structure (no PROCs)



User-Defined Sort

You can incorporate user-defined sorts into any program structure.

For each user-defined sort, the generator builds only an S-1000 section containing a SORT statement that invokes either an input procedure or file and either an output procedure or file (based on what the user entered on the Update Sort Definition screen). You must then construct all the code needed for required input and/or output procedures—including COBOL RELEASE and RETURN statements or PL/I CALL PLIRETC statements—if an input or output procedure is requested.

Like the mainline sort, the user-defined sort supports all four sort formats:

- Sort with USING (input file) and an OUTPUT procedure
- Sort with GIVING (output file) and an INPUT procedure
- Sort with INPUT and OUTPUT procedures
- Sort with USING and GIVING (input and output files)

Generated Sort Objects

A sort program needs to contain the following objects (COBOL and PL/I have slightly different requirements):

SELECT statement (COBOL only)

The sortname must be unique to handle a situation where more than one sort command is specified in a program.

Sort SD (COBOL only)

CA Telon defines an SD for each sort command you desire. (Note that the layout for this sort file does not have to be the same as a corresponding file that is being sorted.)

Sort statement

You define most of the data required to generate the SORT statement on the Update Sort Definition screen in the TDF. CA Telon generates the name of the sort file. For COBOL, if the user does not specify a COPY member for the sort file's definition (and therefore defines sort fields positionally), CA Telon generates in the SD a definition of the file with specified keys.

This is the only sort component that CA Telon generates for user-defined sorts.

SORT INPUT/OUTPUT

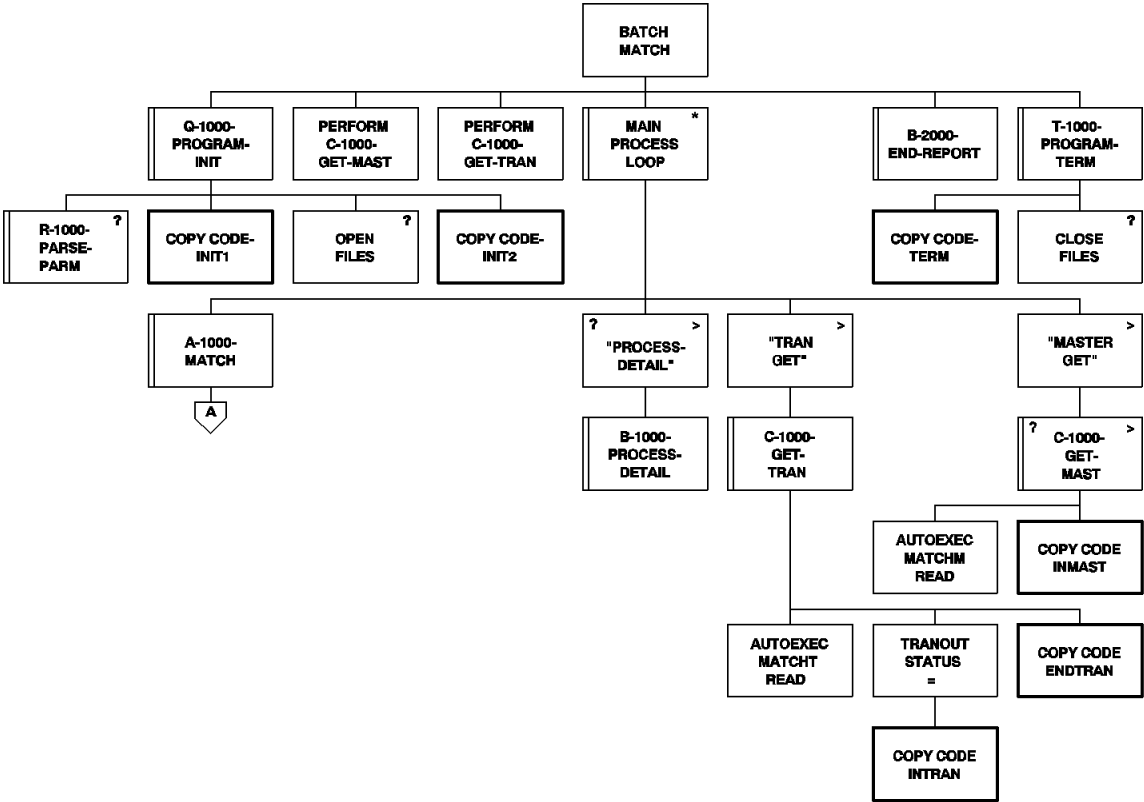
For user-defined sorts only, you are entirely responsible for coding required input and/or output procedures in custom code points referred to in the INPUT and OUTPUT fields on the Update Sort Definition screen. Each of these procedures will consist of one or more paragraphs in custom code that are copied/included in an S-1000 section. The user can call this section from any point in the program, as is done with user I/O data access. If you fail to code the required statements in either procedure (for example, RELEASE in a COBOL input procedure), the generated application program will not run successfully.

Batch Match

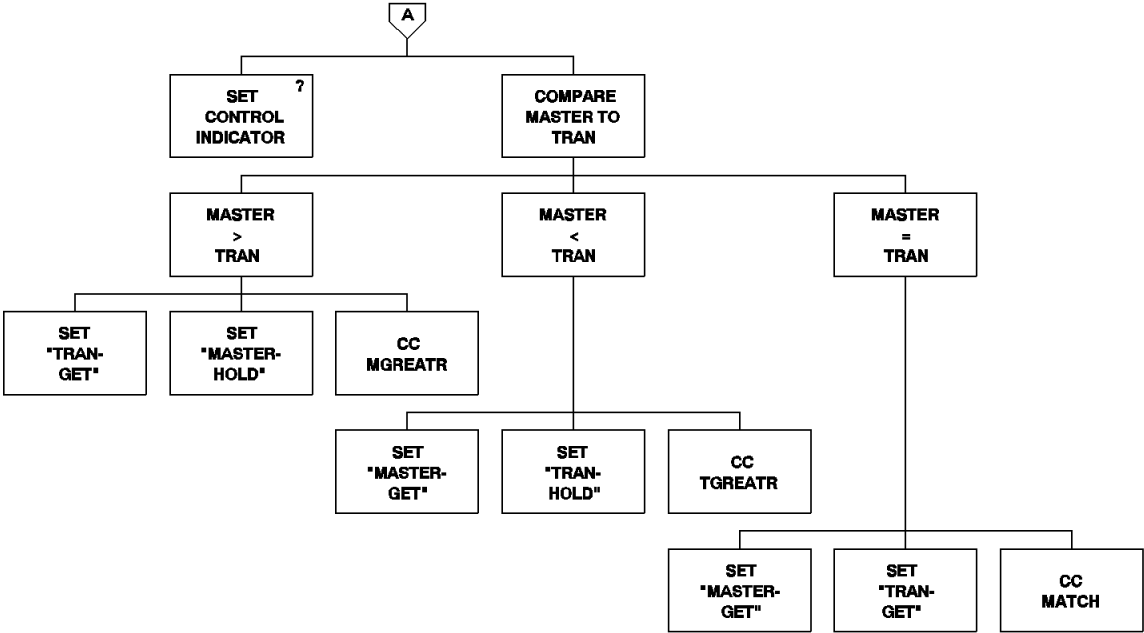
The batch match feature in comparison to the standard mainline programming structure significantly redesigns the mainline section, as well as generating different sections and custom code logic points. The match is limited to two files: transaction (identified by the auto exec request MATCHT) and master (identified by the auto exec request MATCHM).

CA Telon match processing assumes that the master file and transaction file will be processed sequentially (in keyed sequence).

Batch match program structure:



Batch match program structure (continued)



Generated Objects for Match

There are several objects required to drive a match program:

MASTER-INDICATOR

Used to direct PERFORMs of the C-1000-GET-MAST section, which reads master records, or of match custom code logic points. It can have three possible values:

- MASTERGET: requests a read of the next master record
- MASTERHOLD: suppresses performance of the master record read
- MASTEREND: indicates that end-of-file has been reached for the master file

TRAN-INDICATOR

Used to direct PERFORMs of the C-1000-GET-TRAN section, which retrieves transaction records. It can have three possible values:

- TRANGET: requests a read of the next transaction record
- TRANHOLD: suppresses performance of a transaction record read
- TRANEND: indicates that end-of-file has been reached for the transaction file

GETTRAN-INDICATOR

Used to direct internal processing of the C-1000-GET-TRAN section. It can have three possible values:

- ' ' (blank)—Retrieves the first transaction for a key from the holding area
- TRANREAD—Reads the next transaction
- TRANDONE—Indicates that the last transaction for a key has been fetched

TRAN-COUNTER

Keeps track of the number of transaction records read in for the same key.

Match Processing

The principal features of the match program are as follows:

- Mainline structure—Performs set-up processing and then calls the main processing routine until end-of-file is reached for both the master and transaction files.
- C-1000-GET-MAST—The generated code in this section reads a master file record and then calls a generated routine to set up the key. A custom code logic point, INMAST, allows you to perform any necessary processing on each master record before CA Telon performs the matching logic.

- C-1000-GET-TRAN—This section is designed to ensure that you have full flexibility to deal with multiple transactions with the same key value. You can make use of multiple transactions or flag the existence of more than one as an error. A custom code logic point, INTRAN, allows you to direct the destination of a transaction record as soon as it is read, and before it is matched to the master. For example, load it to a buffer, flag it as erroneous.
- C-1000-END-TRAN—This section indicates that end-of-file has been reached for the transaction file. CA Telon uses a custom code logic point, ENDTRAN, after fetching all transactions with a matching key; the first transaction for the next key is stored in a hold area. This allows you to perform any wrap-up processing on a batch of transaction records. There is no way to recognize the last transaction for any key until ENDTRAN is reached.
- A-1000-MATCH—Performs the match of master and transaction keys. The comparison between master and transaction file keys can produce three possible results. Each of these results in the execution of a custom code logic point:
 - MATCH—Executed when the master and transaction keys match. You can insert custom code to process this condition (for example, update master file and write output master file record; or, if the transaction is a DELETE instruction, suppress write of the output master file record). After executing this code, MASTER-INDICATOR is set to MASTERGET, TRAN-INDICATOR to TRANGET, and GETTRAN-INDICATOR to TRANREAD.
 - MGREATR—Executed when the master key is greater than the transaction key (for example, when there is no master for the current transaction record). You can insert custom code here to process this condition (for example, add a new transaction record to the master file, or, if the master-greater-than-transaction condition was not expected, process an error). After executing this code, MASTER-INDICATOR is set to MASTERGET, TRAN-INDICATOR is set to TRANGET, and GETTRAN-INDICATOR is set to TRANREAD.
 - TGREATR—Executed when the transaction key is greater than the master key (for example, when there is no transaction record to match the current master). You can insert custom code here to process this condition (for example, note a missing transaction record, or insert remaining transaction records after the master file has reached end-of-file). After executing this code, TRAN-INDICATOR is set to TRANHOLD, and MASTER-INDICATOR is set to MASTERGET.

Batch Merge

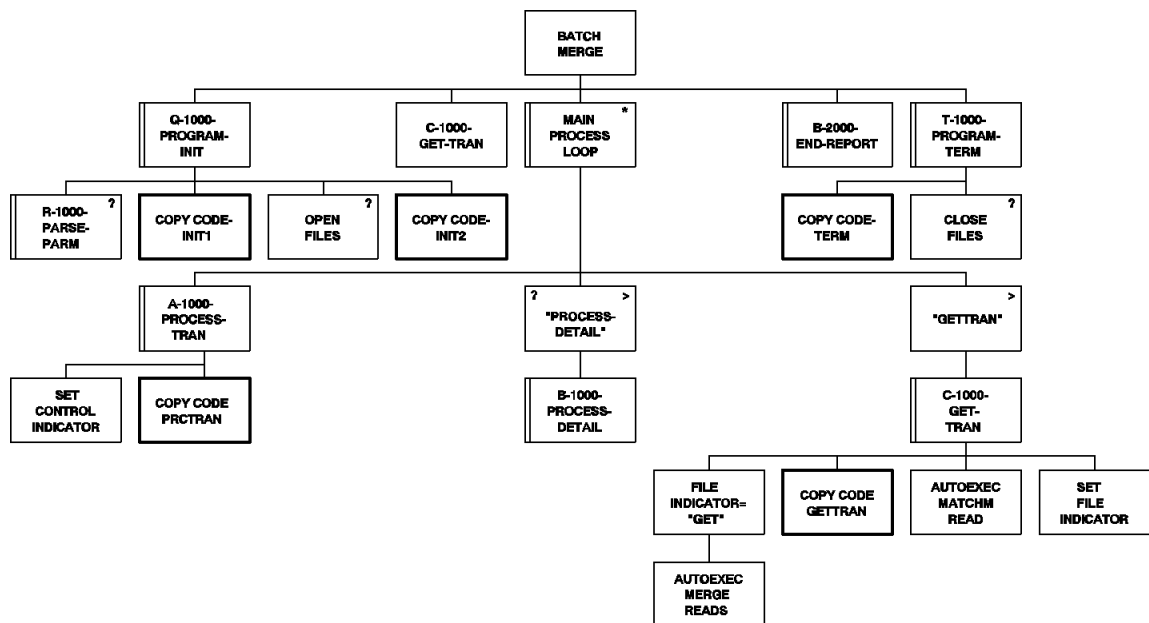
The merge design for the batch system uses the current Batch structure with only a minor modification.

The merge C-1000 section automatically reads a record for any merge file with a FILE-INDICATOR value of GET. Then after the GETTRAN custom code point, CA Telon selects the next record in sequence (*current* record) from the set of input files being merged, handling them in a synchronized way so that only one record at a time is designated as *current*. The A-1000 and B-1000 sections remain unchanged.

You can merge from 2 to 20 files. Merge files are identified by the auto exec requests MERGE01 through MERGE20. Files are identified hierarchically according to the auto exec merge request assigned to each (for example, in the case of matching records, the record from the MERGE01 file takes precedence over the record from the MERGE02 file, and so on).

It is your responsibility to ensure that all files are correctly sorted in the specified key order prior to processing.

Batch merge program structure



Generated Merge Objects

Described below are the objects CA Telon generates to drive a merge program.

FILE-INDICATOR-TABLE

CA Telon sets up a table with an entry for each file to keep track of its current status. Valid values for these table entries are as follows:

- GET—On the next pass through C-1000, fetch a record from this file
- HOLD—On the next pass through C-1000, do NOT fetch a record from this file
- END—End-of-file has been reached for this input file

MERGE-KEY-TABLE

Contains the current key values for each of the input files. These are used to determine what is the *current* record in merge processing. Values are compared only for files that have not reached end-of-file. The row length for the MERGE-KEY-TABLE will be the sum of the key lengths.

MIN-PTR

Keeps track of the lowest key (and therefore the *current* record) among the current key values.

Merge Processing

C-1000 merge processing is done as follows:

1. For each input file, a record is read if the value of the FILE-INDICATOR for that file is *GET*. (For the first iteration of C-1000, all FILE-INDICATORS are set to *GET*.) When end-of-file is reached for any file, *END* is moved to its FILE-INDICATOR entry.
2. Users can insert custom code into GETTRAN to manipulate keys or perform any other required custom processing.
3. Generated code sets up the key for each file with an indicator and moves it to the file's MERGE-KEY-TABLE row.
4. Generated code determines the lowest key among the merge files, and points MIN-PTR to it. If there are matching keys the first of the matching keys (as described above) is considered the lowest. The FILE-INDICATOR for the file with the lowest key is set to *GET*, and indicators for the other files (that have not reached end-of-file) are set to *HOLD* to suppress the read on the next pass.
5. When all files have reached end-of-file (for example, their FILE-INDICATORS are set to *END*) at the end of C-1000 processing, the batch CONTROL-INDICATOR is set to *END-TRAN-LIT*.

Program Control

Program flow in all CA Telon batch programs defaults to a continuous loop of C-1000-GET-TRAN, A-1000-PROCESS-TRAN, and B-1000-PROCESS-DETAIL (groupname) until no more input transactions exist for the program. You can control and override this default flow by using the GETTRAN parameter, the PRCTRAN parameter, or a combination of both of these COPY code exits. The following text discusses these two exits individually and shows common uses for each of them.

BATCH/GETTRAN

The GETTRAN COPY code exit is included in program section C-1000-GET-TRAN and is specified on the Create/Update Batch Definition screen. GETTRAN allows for any custom code required to obtain a transaction which the output side of the batch program can use. You can include user-written loops here to read multiple records when specific search criteria must be met for report processing.

If IMS checkpoint/restart is required in the batch program, custom code that you insert in C-1000 should first issue a CHKP call to create a restart point between the processing of one transaction and the obtaining of a new one.

BATCH/PRCTRAN

The PRCTRAN COPY code exit, as specified on the Create/Update Batch Definition screen, is included in program section A-1000 PROCESS-TRAN after the default selection of the first detail group defined for output. You can use custom code that you insert in this section to produce multiple detail groups of output by setting the CONTROL-INDICATOR (discussed earlier) to the group desired and performing B-1000-PROCESS-DETAIL (groupname) for as many groups as need be printed. After performing B-1000-PROCESS, CA Telon sets the CONTROL-INDICATOR to get the next transaction. The CONTROL-INDICATOR must be set by means of custom code for each time B-1000 performs.

CICS Nonterminal Program Structure

CA Telon supports the new CICS nonterminal program structure (shown in the following diagram), along with CICS queues and journals (for both screen and nonterminal CICS programs), by using new parameters entered on the CA Telon Design Facility and converted into CA Telon source code by the Export Facility.

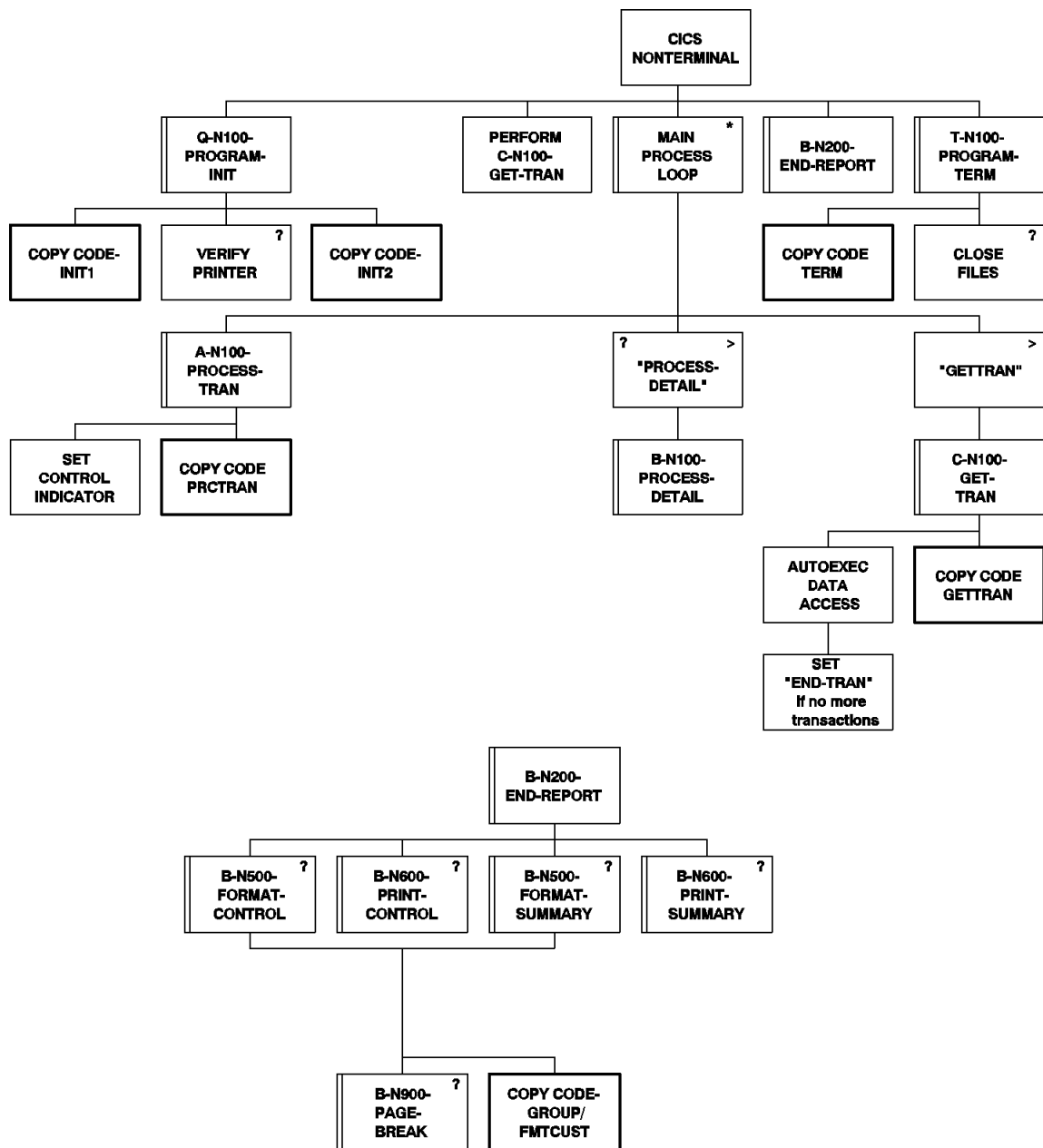
CICS nonterminal program structure is similar to that of standard batch programs. The principal differences between the batch or the CICS screen program structures and the CICS nonterminal program structure are:

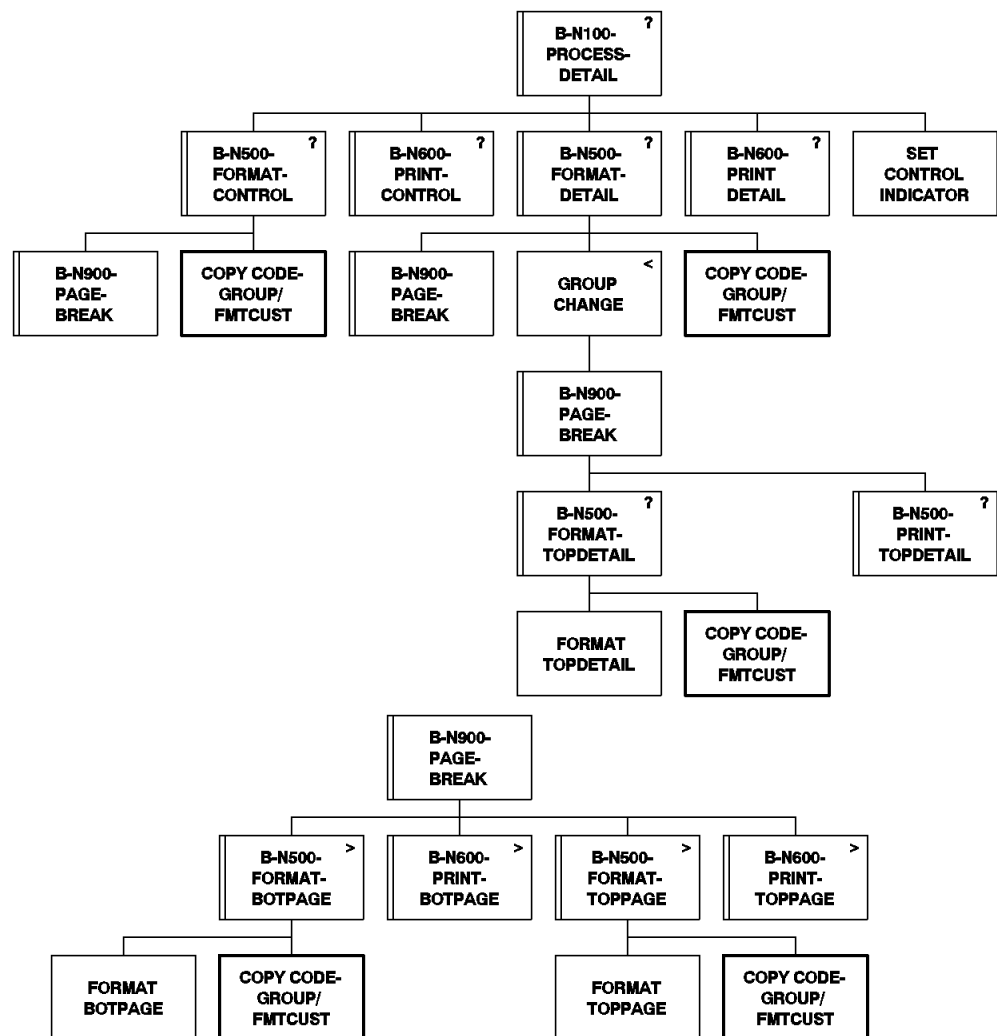
1. All section names contain -N100 rather than -1000, following the online naming convention rather than the batch.
2. There is a Q-N100 section code where you determine printer destination. If the RPTDEST parameter contains the keyword PRINTER, CA Telon generates code here to check that the printer identified by the variable BWA-PRINTER-ID (COBOL) or BWA_PRINTER_ID (PL/I) exists in the TCT. This check ensures that at run time this variable is defaulted to the value of the PRTDEST parameter.

PRTDEST is required with PRINTER. If the RPTDEST is a VSAM file, the file named must also be defined in the program's data group.

3. When an AUTOEXEC TRANSACT is requested, the C-N100 section does the following:
 - a. Performs *establish position* or *first record* logic, if needed (for example, VSAM STARTBR, DB2 OPEN CURSOR, DL/I GU), for the first transaction (if BWA-TRANSACTION-COUNT = 1).
 - b. Performs *read next item* (for example, VSAM READNEXT, DB2 FETCH, DL/I GN) to retrieve the next transaction, except when first record logic is performed.

- c. Performs closing logic, if needed (for example, VSAM ENDBR, DB2 CLOSE CURSOR), after last transaction (if END-TRAN).





4. If you specify a printer, C-N200 section code does the following:
 - a. Collects a page of print data (as defined by the SIZE parameter).
 - b. Envelopes the page of data with the correct control characters for the requested printer (see number 2 earlier in this topic) when a logical page break is reached.
 - c. Sends the data to the printer.
5. The T-N100 section performs no processing other than the code inserted by the user in custom code point TERM (for example, CLOSE FILE logic generated in the batch counterpart, T-N100, is not generated).

6. Data Access I/O requests in CICS nonterminal programs match those currently generated for screen-oriented CICS programs (for example, "EXEC CICS...").
7. Working storage variable names are a mix of the current CICS and batch names; those associated with report production maintain the batch prefix BWA, and those related to data types and I/O requests begin with SYSWK.

CICS Client Program Hierarchical Charts

This subsection discusses the CICS Client program hierarchical charts in terms of a program overview, main processing, output processing, and input processing.

Program Overview

A generated CICS Client program is logically broken into two main parts: an output part, which reads and formats data to be presented on the screen, and an input part, which processes the terminal user's action (for example, PF key pressed, data entered, and so on). In addition, a CICS Client program may contain other components related to the teleprocessing environment it runs under (for example, the transaction scheduling for CICS). This chapter presents only the output and input parts of the CICS Client program regardless of the teleprocessing options in effect.

CONTROL-INDICATOR

Before you can fully understand program flow, you must know the function of the CONTROL-INDICATOR. The CONTROL-INDICATOR is a program field that specifies the next action that the program is to perform. In most cases, the CONTROL-INDICATOR is set automatically by CA Telon; however, you can alter the value of the CONTROL-INDICATOR in custom code. The following table illustrates the CONTROL-INDICATOR settings:

Setting	LIT Data Item	Logic
O	PROCESS-OUTPUT-LIT	PROCESS-OUTPUT
I	PROCESS-INPUT-LIT	PROCESS-INPUT
R	DO-TRANSFER-LIT	DO-TRANSFER
E	DO-WRITE-LIT	DO-WRITE
C	TRANSACTION-COMPLETE-LIT	TRANSACTION- COMPLETE
space	CONTINUE-PROCESS-LIT	CONTINUE-PROCESS

To alter the program flow using custom code, you must set the CONTROL-INDICATOR to the appropriate value, then branch to the return label of the current section. The following example illustrates how custom code sends an error message to the screen and transfers control to another program.

- To send an error message to the screen: Set the error message and any highlight attributes, MOVE DO-WRITE-LIT to CONTROL-INDICATOR, and GO TO end-of-routine. For example:

```
MOVE 'ERROR-MESSAGE' TO TP0-ERRMSG1
MOVE ERROR-ATTR TO TP0-FIELD-ATTR
MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
GOTO B-100-OUTPUT-INIT-RETURN
```

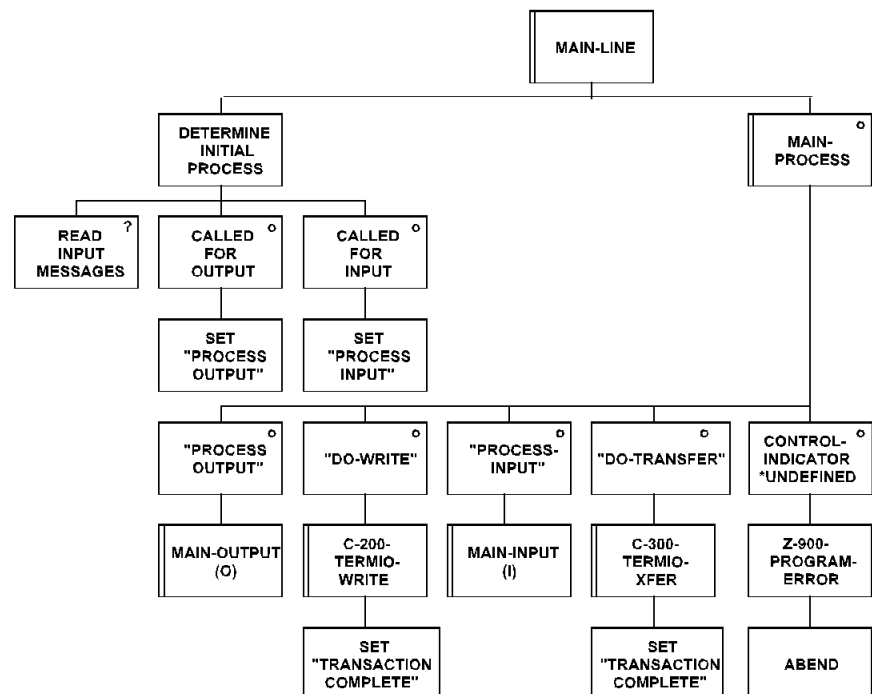
- To transfer control to another program:

MOVE 'next-program-id' to NEXT-PROGRAM-NAME-ID, MOVE DO-TRANSFER-LIT to CONTROL-INDICATOR, and GO TO end-of-routine. For example:

```
MOVE 'XXXX' TO NEXT-PROGRAM-NAME-ID
MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
GOTO P-100-PFKEYS-RETURN
```

Main Processing

The major activities, shown in the hierarchical chart for mainline processing, are: PROCESS-OUTPUT, PROCESS-INPUT, DO-WRITE, and DO-TRANSFER.



In IMS dynamic link programs, a READ (of the IMS SPA) is performed before entry into the program MAIN-LINE. In order to prevent a wild branch into the MAIN-LINE from the Z-980-ABNORMAL-TERM section (which would occur if the READ failed), the CONTROL-INDICATOR is set to 'Z' ("INITIAL-PROCESS") in both the IMS-PRIMARY-ENTRY section and the IMS-PRIMARY-ENTRY-PROCESS sections before the READ.

If the PRIMARY-ENTRY sections execute normally, execution proceeds. If an abend occurs in either section, the Z-980-ABNORMAL-TERM section is executed, and control is returned to the section where the abend occurred.

MAIN-LINE

In MAIN-LINE processing, CA Telon first performs the DETERMINE INITIAL PROCESS routine to determine whether to set the CONTROL-INDICATOR to PROCESS-INPUT or PROCESS-OUTPUT.

The MAIN-PROCESS routine performs one of the following routines on each iteration depending on the value of the CONTROL-INDICATOR. Once MAIN-PROCESS performs a routine, the value of the CONTROL-INDICATOR is automatically set to specify the next action that the program is to perform. The following table illustrates how the CONTROL-INDICATOR is set:

Set to	Meaning	Next ROUTINE performed
O	PROCESS-OUTPUT	MAIN-OUTPUT to process the output side.
I	PROCESS-INPUT	MAIN-INPUT to process the input side.
R	DO-TRANSFER	C-300-TERMIO-XFER to transfer to the next screen.
E	DO-WRITE	C-200-TERMIO-WRITE to write a screen.
other values	undefined	ABEND.

As you can see from the above table, the flow of control between PROCESS-OUTPUT, PROCESS-INPUT, DO-TRANSFER, and DO-WRITE is controlled by the CONTROL-INDICATOR field.

The first action you or the application user takes upon designing or entering the program is to determine whether PROCESS-OUTPUT or PROCESS-INPUT is to be performed first. The method of determining which process to perform varies for each environment. Once you determine which process is to be executed, the CONTROL-INDICATOR is automatically set accordingly. The MAIN-PROCESS routine then performs until the CONTROL-INDICATOR is set to TRANSACTION-COMPLETE.

MAIN-PROCESS

Within MAIN-PROCESS, one of the following routines is performed:

- PROCESS-OUTPUT
- DO-WRITE (to perform C-200-TERMIO-WRITE)
- PROCESS-INPUT
- DO-TRANSFER (to perform C-300-TERMIO-XFER)
- CONTROL-INDICATOR *UNDEFINED (to perform an ABEND routine)

As noted earlier, the flow of processing depends on the value of the CONTROL-INDICATOR. For example, when the CONTROL-INDICATOR is set to a value of I (PROCESS-INPUT-LIT), the MAIN-INPUT routine performs. This routine processes the input side of the transaction. The program tests the CONTROL-INDICATOR for a value of space (CONTINUE-PROCESS-LIT) before performing each routine. If all routines are performed and the CONTROL-INDICATOR is still set to space (" "), the CONTROL-INDICATOR is automatically set to R (DO-TRANSFER-LIT). This is the default action after processing input.

DO-TRANSFER performs the C-300-TERMIO-XFER routine to transfer control to the next program (specified by the NEXTPGM parameter in your program definition). The default action after DO-TRANSFER is to SET 'TRANSACTION COMPLETE'. This ends the MAIN-PROCESS routine.

When the CONTROL-INDICATOR is set to O (PROCESS-OUTPUT-LIT), the program performs the MAIN-OUTPUT routine. This routine processes the output side of the transaction. The program tests the CONTROL-INDICATOR for a value of space (CONTINUE-PROCESS-LIT) before performing each routine. As on the INPUT side, if the CONTROL-INDICATOR is still set to space (" ") when all the routines have been performed, the CONTROL-INDICATOR is automatically set to E (DO-WRITE-LIT). This is the default action after processing output.

DO-WRITE performs C-200-TERMIO-WRITE to send a message to the terminal. The default action after DO-WRITE is to SET 'TRANSACTION COMPLETE'. This ends the MAIN-PROCESS routine.

Output Processing

The CICS Client program hierarchical chart illustrates output processing. In output processing, the MAIN-OUTPUT (O) routine controls output processing for a transaction. Each subsequent routine is performed only if the CONTROL-INDICATOR is space (" "). The default action at the end of MAIN-OUTPUT is SET 'DO-WRITE'. The Output Process may consist of: K-100-HOLD-RESTORE , C-500-FORM-INIT, and SET 'DO-WRITE' (to end Output Processing).

K-100-HOLD-RESTORE

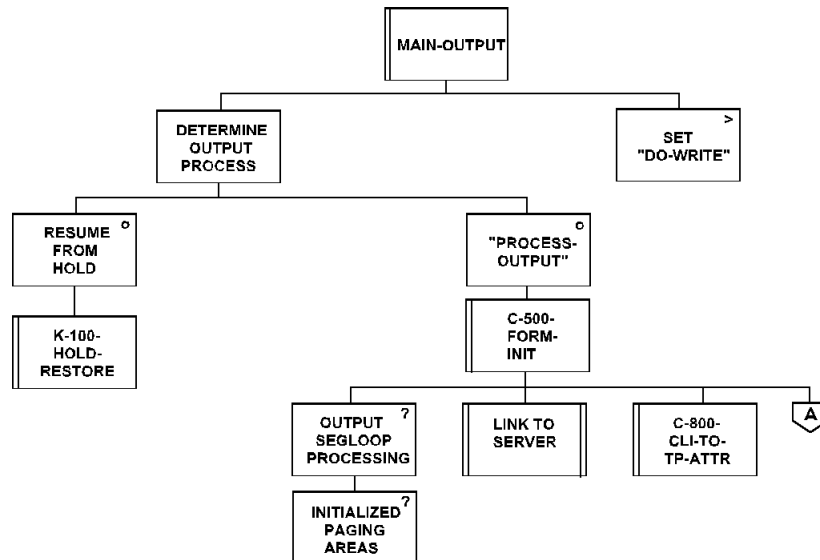
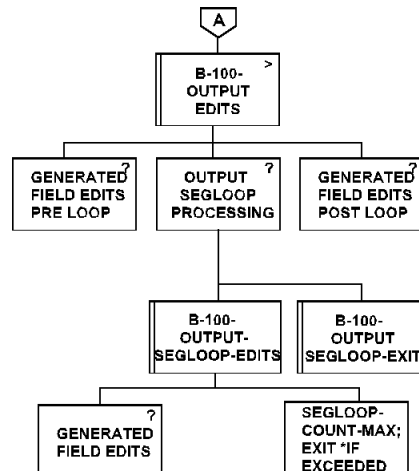
CA Telon generates K-100-HOLD-RESTORE only if HOLD=Y or HELP=Y on the Update Screen Parameters screen, or HELPMMSG is indicated on the Update HELPMMSG Parameter screen, Update Output, Input, Outin Fields screen, or the Update Select Fields screen.

C-500-FORM-INIT

C-500-FORM-INIT is performed to accomplish the following three objectives: (1) prepare for the Link to the Server program, (2) Link to the Server program, and (3) process information passed from the Server program. There are differences in generated code for this section, depending on whether or not there is an output SEGLOOP specified.

When there is no output SEGLOOP, the SPA-PROCESS-REQUEST indicator is set to '1' to cause the Server to perform FORM-INIT processing. Then a Link is performed to the Server. Upon return, the C-800-CLI-TO-TP-ATTR section is performed to map field attributes from their one-byte Server value to the three- or six-byte Client (i.e., screen) value for CICS. The final step is to perform B-100-OUTPUT-EDITS to perform the corresponding mapping of the field values from the DBNAMEs to the TPO- values.

When there *is* a SEGLOOP, the B-100-OUTPUT-SEGLOOP-INIT paragraph is performed, and other logic is generated dealing with SEGLOOP paging and browse keys. Then a Link is performed to the Server. Upon return, the C-800-CLI-TO-TP-ATTR section is performed to map field attributes from their one-byte Server value to the three- or six-byte Client (i.e., screen) value for CICS. The final step is to perform B-100-OUTPUT-EDITS. Note that the B-100-OUTPUT-EDITS code that is generated for the CICS Client does NOT access custom code nor any data access. It simply deals with mapping from the DBNAMEs to the TPO- values.

C-500-FORM-INIT hierarchical chart**C-500-FORM-INIT hierarchical chart (continued)****B-100-OUTPUT-EDITS**

The B-100-OUTPUT-EDITS section handles editing and formatting of all OUTPUT/OUTIN fields. Additionally, it contains the code generated when an OUTPUT/OUTIN SEGLOOP is required.

The GENERATED FIELD EDITS *function* contains code specified by the FLDTYPE parameter on the Update Panel Fields screen (for example, to edit any OUTPUT or OUTIN fields).

OUTPUT SEGLOOP PROCESSING (SEGLOOP Mapping) exists only if there is an OUTPUT or OUTIN type SEGLOOP specified in the program definition. When SEGLOOP Mapping exists, it includes B-100-OUTPUT-SEGLOOP-EDITS, and B-100-OUTPUT-SEGLOOP-EXIT. These are generated in the order specified.

Note that:

- During execution, B-100-OUTPUT-SEGLOOP-EDITS is repeated zero to n times, where n is the number of groups in the SEGLOOP (as determined by the INCRE parameter on the Create/Update Table SEGLOOP screen). When the repetition is complete, the box following the B-100-OUTPUT-SEGLOOP-EDITS gains control. (Specifically, this is SET 'DO-WRITE'.)

To process each group of fields in the loop, the program executes GENERATED FIELD EDITS through the SEGLOOP-COUNT-MAX check where SEGLOOP-COUNT is compared with a specified maximum value.

- GENERATED FIELD EDITS exists unless all of the OUTPUT or OUTIN fields specify FLDTYPE=**NONE** on the Update Panel Fields screen, Update Output, Input, Outin Fields screen, or Update Select Field screen. It is possible, but very unusual, that GENERATED FIELD EDITS will not exist.

Edits are performed for any fields defined after the loop.

- B-100-OUTPUT-SEGLOOP=EXIT exits from OUTPUT SEGLOOP PROCESSING.

SET 'DO-WRITE'

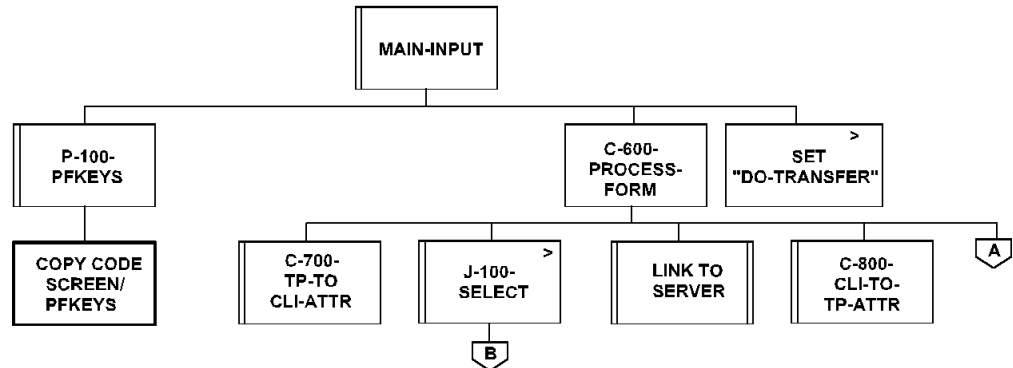
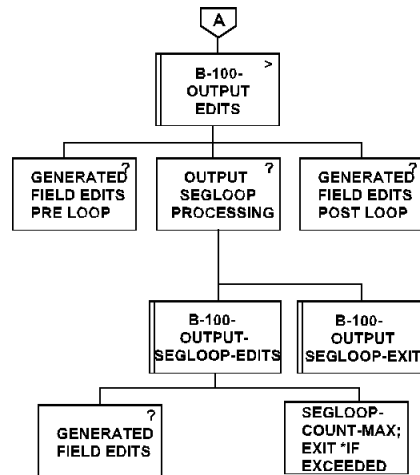
Finally, SET 'DO-WRITE' gains control and writes the screen to the terminal, thereby completing MAIN-OUTPUT.

At this point, output processing is complete and input processing of the screen can begin.

Input Processing

The input process, as shown in hierarchical chart for input processing, consists of P-100-PFKEYS, C-600-PROCESS-FORM, and a set DO TRANSFER.

Note: If an error is detected during input processing, an error message is sent to the application user's terminal.

Input process hierarchical chart**Input process hierarchical chart (continued)****P-100-PFKEYS**

P-100-PFKEYS comprises the possibility of PF-key processing inherent to the screen (such as paging PF keys for list processing). If you define a SEGLOOP, CA Telon automatically creates this section in your program.

P-100-PFKEYS is followed by the COPY code requested by the PFKEYS parameter on the Create/Update Screen Definition screen.

C-600-FORM-PROCESS

C-600-FORM-PROCESS is performed to accomplish the following three objectives: (1) prepare for the Link to the Server program, (2) Link to the Server program, and (3) process information passed from the Server program. There are differences in generated code for this section, depending on whether or not there is an output SEGLOOP specified.

Prior to the Link to the Server, the C-700-TP-TO-CLI-ATTR section is performed to convert CICS field attributes to their Server one-byte counterparts. Then, J-100-SELECT is performed to determine if any SELECT conditions were activated during PFKey processing. Should that be the case, and there be some modification to the CONTROL-INDICATOR, it is possible that the Link to the Server could be bypassed.

If the result of J-100-SELECT processing is CONTINUE-PROCESS, the SPA-PROCESS-REQUEST indicator is set to '0' to cause the Server to perform FORM-PROCESS processing. Next, a Link is performed to the Server. Upon return, the C-800-CLI-TO-TP-ATTR section is performed to map field attributes from their one-byte Server value to the three- or six-byte Client (i.e., screen) value for CICS. The final step is to perform B-100-OUTPUT-EDITS to perform the corresponding mapping of the field values from the DBNAMEs to the TPO- values.

J-100-SELECT

The J-100-SELECT processing involves the following: PERFORM M-100 FOR FIELD HELP, DETERMINE OPTION SELECTED, and J-100-SELECT OPT-fldname.

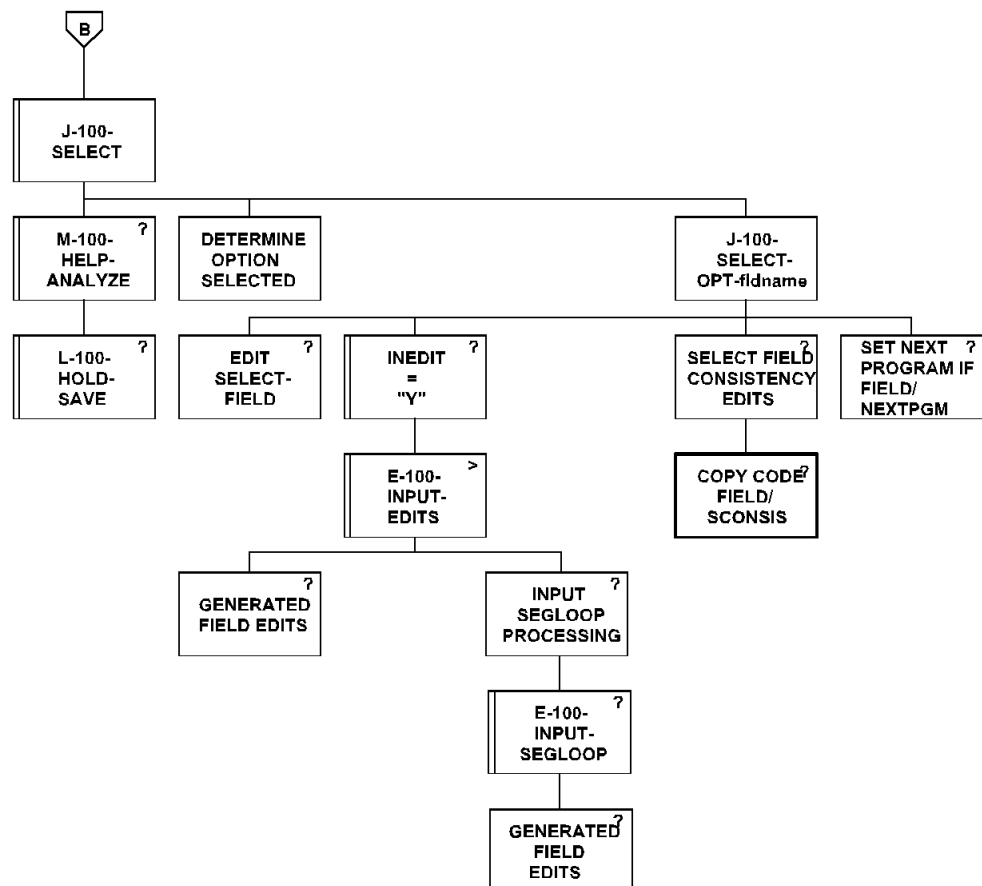
J-100-SELECT contains the following components:

- PERFORM M-100 FOR FIELD HELP , which exists if HELP=**Y** on the Update Screen Parameters screen.
- DETERMINE OPTION SELECTED

- J-100-SELECT-OPT-fldname , which exists if a non-blank value was entered for the SELECT FIELDS parameter on the Create/Update Screen Definition screen.

Following J-100-SELECT-OPT-fldname are EDIT-SELECT-FIELD, PERFORM E-100 generated FIELD/INEDIT logic, SELECT FIELD CONSISTENCY EDITS, and SET NEXT PROGRAM NEXTPGM logic. Note that all of this logic is conditional.

- EDIT-SELECT-FIELD exists based on specifications entered on the Panel Definition menu.
 - The E-100 custom code member specified by the FLDEDIT parameter on the Create/Update Screen Definition screen performs at this point if INEDIT=**Y** was specified on either the Update Select Field screen or the Update Select Parameters screen.
 - SELECT FIELD CONSISTENCY EDITS exists only if the SCON SIS parameter is specified on either the Update Select Field screen or the Update Select Parameters screen.
- SET NEXT PROGRAM NEXTPGM logic exists only if the NEXTPGM parameter was specified on the Update Select Field screen.



SET 'DO-TRANSFER'

SET 'DO-TRANSFER' performs processing to transfer control to the next screen.

CICS Server Program Hierarchical Charts

This subsection discusses the CICS Server program hierarchical charts in terms of a program overview, main processing, output processing, and input processing.

Program Overview

A generated CICS Server program is logically broken into four main parts: (1) INIT-FORM, which reads and formats data to be passed to the Client program prior to the first display of the screen or form, (2) PROCESS-FORM, which processes the user specifications accepted by the Client program for the entire screen or form, (3) PROCESS-FIELD, which processes user specifications accepted by the Client program for an individual field on the screen or form, and (4) TERM-FORM, which concludes processing for the screen or form.

CONTROL-INDICATOR

Before you can fully understand program flow, you must know the function of the CONTROL-INDICATOR. The CONTROL-INDICATOR is a program field that specifies the next action that the program is to perform. In most cases, the CONTROL-INDICATOR is set automatically by CA Telon; however, you can alter the value of the CONTROL-INDICATOR in custom code. The following table illustrates the CONTROL-INDICATOR settings:

Setting	LIT Data Item	Logic
O	INIT-FORM-LIT	INIT-FORM
E	DO-WRITE-LIT	DO-WRITE
F	PROCESS-FIELD-LIT	PROCESS-FIELD
I	PROCESS-FORM-LIT	PROCESS-FORM
T	TERM-FORM-LIT	TERM-FORM
C	TRANSACTION-COMPLETE-LIT	TRANSACTION-COMPLETE
space	CONTINUE-PROCESS-LIT	CONTINUE-PROCESS

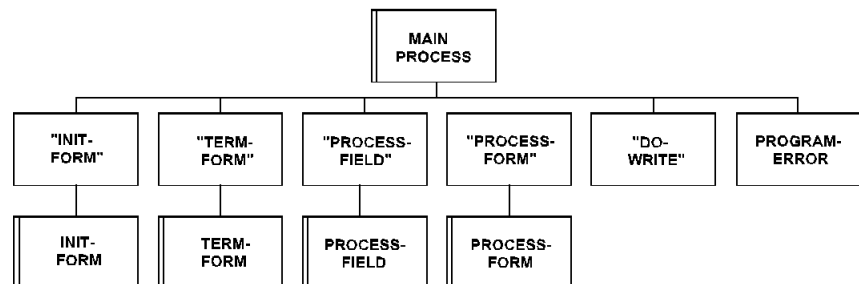
To alter the program flow using custom code, you must set the CONTROL-INDICATOR to the appropriate value, then branch to the return label of the current section. The following example illustrates how custom code sends an error message to the Client program.

- To send an error message to the screen: Set the error message and any highlight attributes, MOVE DO-WRITE-LIT to CONTROL-INDICATOR, and GO TO end-of-routine. For example:

```
MOVE 'ERROR-MESSAGE' TO TP0-ERRMSG1
MOVE ERROR-ATTR TO TP0-FIELD-ATTR
MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
GOTO B-100-OUTPUT-INIT-RETURN
```

Main Processing

The major activities, shown in the hierarchical chart for mainline processing, are: INIT-FORM, PROCESS-FORM, PROCESS-FIELD, TERM-FORM, and DO-WRITE.



IMS-PRIMARY-ENTRY

In IMS dynamic link programs, a READ (of the IMS SPA) is performed before entry into the program MAIN-LINE. In order to prevent a wild branch into the MAIN-LINE from the Z-980-ABNORMAL-TERM section (which would occur if the READ failed), the CONTROL-INDICATOR is set to 'Z' ("INITIAL-PROCESS") in both the IMS-PRIMARY-ENTRY section and the IMS-PRIMARY-ENTRY-PROCESS sections before the READ.

If the PRIMARY-ENTRY sections execute normally, execution proceeds. If an abend occurs in either section, the Z-980-ABNORMAL-TERM section is executed, and control is returned to the section where the abend occurred.

MAIN-LINE

In MAIN-LINE processing, CA Telon first performs the DETERMINE INITIAL PROCESS routine to determine whether to set the CONTROL-INDICATOR to INIT-FORM, PROCESS-FORM, PROCESS-FIELD or TERM-FORM.

The MAIN-PROCESS routine performs one of the following routines on each iteration depending on the value of the CONTROL-INDICATOR. Once MAIN-PROCESS performs a routine, the value of the CONTROL-INDICATOR is automatically set to specify the next action that the program is to perform. The following table illustrates how the CONTROL-INDICATOR is set:

Set to	Meaning	Next ROUTINE performed
O	INIT-FORM	MAIN-FORM-INIT to initialize data for the Client.
I	PROCESS-FORM	MAIN-FORM-PROCESS to process Client input for the entire form.
F	PROCESS-FIELD	MAIN-FIELD-PROCESS to process Client input for one field.
O	TERM-FORM	MAIN-FORM-TERM to complete form processing.
E	DO-WRITE	C-200-CLIENT-WRITE to transfer back to the Client.
other values	undefined	ABEND.

As you can see from the above table, the flow of control between INIT-FORM, PROCESS-FORM, PROCESS-FIELD, TERM-FORM, and DO-WRITE is controlled by the CONTROL-INDICATOR field.

The first action you or the application user takes upon designing or entering the program is to determine whether PROCESS-OUTPUT or PROCESS-INPUT is to be performed first. The method of determining which process to perform varies for each environment. Once you determine which process is to be executed, the CONTROL-INDICATOR is automatically set accordingly. The MAIN-PROCESS routine then performs until the CONTROL-INDICATOR is set to TRANSACTION-COMPLETE.

MAIN-PROCESS

Within MAIN-PROCESS, one of the following routines is performed:

- INIT-FORM
- PROCESS-FORM
- PROCESS-FIELD
- TERM-FORM
- DO-WRITE (to perform C-200-CLIENT-WRITE)
- CONTROL-INDICATOR *UNDEFINED (to perform an ABEND routine)

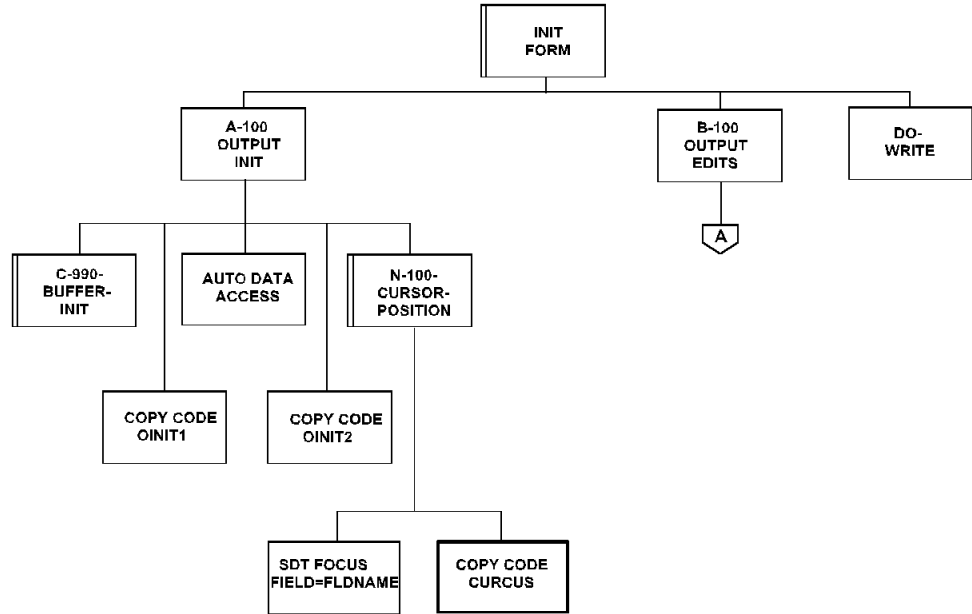
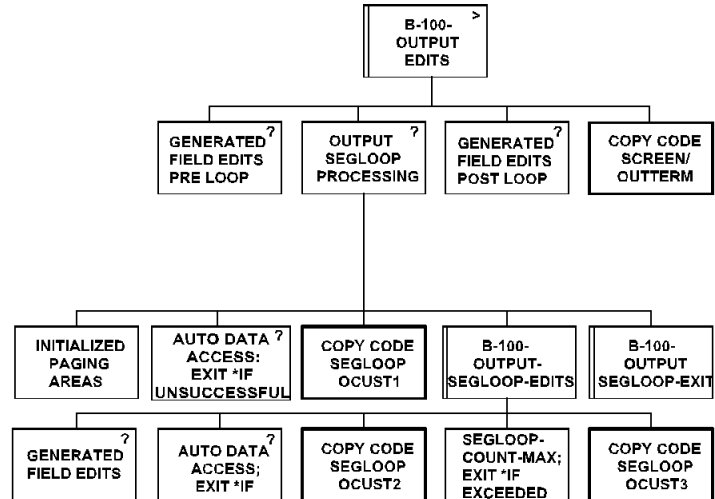
As noted earlier, the flow of processing depends on the value of the CONTROL-INDICATOR. For example, when the CONTROL- INDICATOR is set to a value of I (PROCESS-FORM-LIT), the MAIN- FORM-PROCESS routine performs. This routine processes the input for the form passed from the Client. The program tests the CONTROL-INDICATOR for a value of space (CONTINUE-PROCESS-LIT) before performing each routine. If all routines are performed and the CONTROL-INDICATOR is still set to space (" "), the CONTROL-INDICATOR is automatically set to **E** (DO- WRITE-LIT). This is the default action after processing input.

DO-WRITE performs the C-200-CLIENT-WRITE routine to transfer control back to the Client program.

When the CONTROL-INDICATOR is set to **O** (INIT-FORM-LIT), the program performs the MAIN-FORM-INIT routine. This routine processes data to be passed back to the Client program prior to the first display of the form. The program tests the CONTROL-INDICATOR for a value of space (CONTINUE-PROCESS- LIT) before performing each routine. As with PROCESS-FORM side, if the CONTROL-INDICATOR is still set to space (" ") when all the routines have been performed, the CONTROL-INDICATOR is automatically set to **E** (DO-WRITE-LIT). This is the default action after form initialization.

Form Initialization Processing

The CICS Server program hierarchical chart illustrates form initialization processing. In form initialization processing, the MAIN-FORM-INIT (O) routine controls form initialization processing for a transaction. Each subsequent routine is performed only if the CONTROL-INDICATOR is space (" "). The default action at the end of MAIN-FORM-INIT is SET 'DO-WRITE'. The Form Initialization Process may consist of: A-100-OUTPUT-INIT, B-100-OUTPUT-EDITS, and SET 'DO-WRITE' (to end Form Initialization Processing).

CICS Server program hierarchical chart**CICS Server program hierarchical chart (continued)**

A-100-OUTPUT-INIT

A-100-OUTPUT-INIT comprises any combination of three optional *functions*, and the mandatory cursor-positioning function, as follows: COPY code requested by the OINIT1 parameter, possibly followed by AUTO DATA ACCESS, followed by COPY code requested by the OINIT2 parameter, followed by N-100 CURSOR-POSITION.

The OINIT1 and OINIT2 parameters are specified on the Create/Update Screen Definition screen.

Note: There is no question mark in the upper-right corner of these COPY code boxes. In this case, the use of a question mark would be redundant because, by definition, the code's existence is contingent on whether you include it or not.

The AUTO DATA ACCESS exists if you set up automatic data access on the Create/Update Data Group screen (and for the REQUEST parameter at least one OUTREAD, OIREAD, or UPDATE value is set to **Y**).

N-100-CURSOR-POSITION consists either of COPY Code requested using the CURSCUS parameter on the Create/Update Screen Definition screen, or logic generated to position the cursor to the screen, as defined using the CURSOR parameter on the Create/Update Screen Definition screen. For any single program, you can only specify one of these parameters.

B-100-OUTPUT-EDITS

The B-100-OUTPUT-EDITS section handles editing and formatting of all OUTPUT/OUTIN fields. Additionally, it contains the code generated when an OUTPUT/OUTIN SEGLOOP is required.

The GENERATED FIELD EDITS *function* contains code specified by the FLDTYPE parameter on the Update Panel Fields screen (for example, to edit any OUTPUT or OUTIN fields).

OUTPUT SEGLOOP PROCESSING (SEGLOOP Mapping) exists only if there is an OUTPUT or OUTIN type SEGLOOP specified in the program definition. When SEGLOOP Mapping exists, it includes B-100-OUTPUT-SEGLOOP-EDITS, and B-100-OUTPUT-SEGLOOP-EXIT. These are generated in the order specified. Note that:

- During execution, B-100-OUTPUT-SEGLOOP-EDITS is repeated zero to n times, where n is the number of groups in the SEGLOOP (as determined by the INCRE parameter on the Create/Update Table SEGLOOP screen). When the repetition is complete, the box following the B-100-OUTPUT-SEGLOOP-EDITS gains control. (Specifically, this is SET 'DO-WRITE'.)

To process each group of fields in the loop, the program executes GENERATED FIELD EDITS through the SEGLOOP-COUNT-MAX check where SEGLOOP-COUNT is compared with a specified maximum value.

- GENERATED FIELD EDITS exists unless all of the OUTPUT or OUTIN fields specify FLDTYPE=**NONE** on the Update Panel Fields screen, Update Output, Input, Outin Fields screen, or Update Select Field screen. It is possible, but very unusual, that GENERATED FIELD EDITS will not exist.

Edits are performed for any fields defined after the loop.

- B-100-OUTPUT-SEGLOOP=EXIT exits from OUTPUT SEGLOOP PROCESSING.

OUTTERM, if specified on the Create/Update Screen Definition, is processed next.

SET 'DO-WRITE'

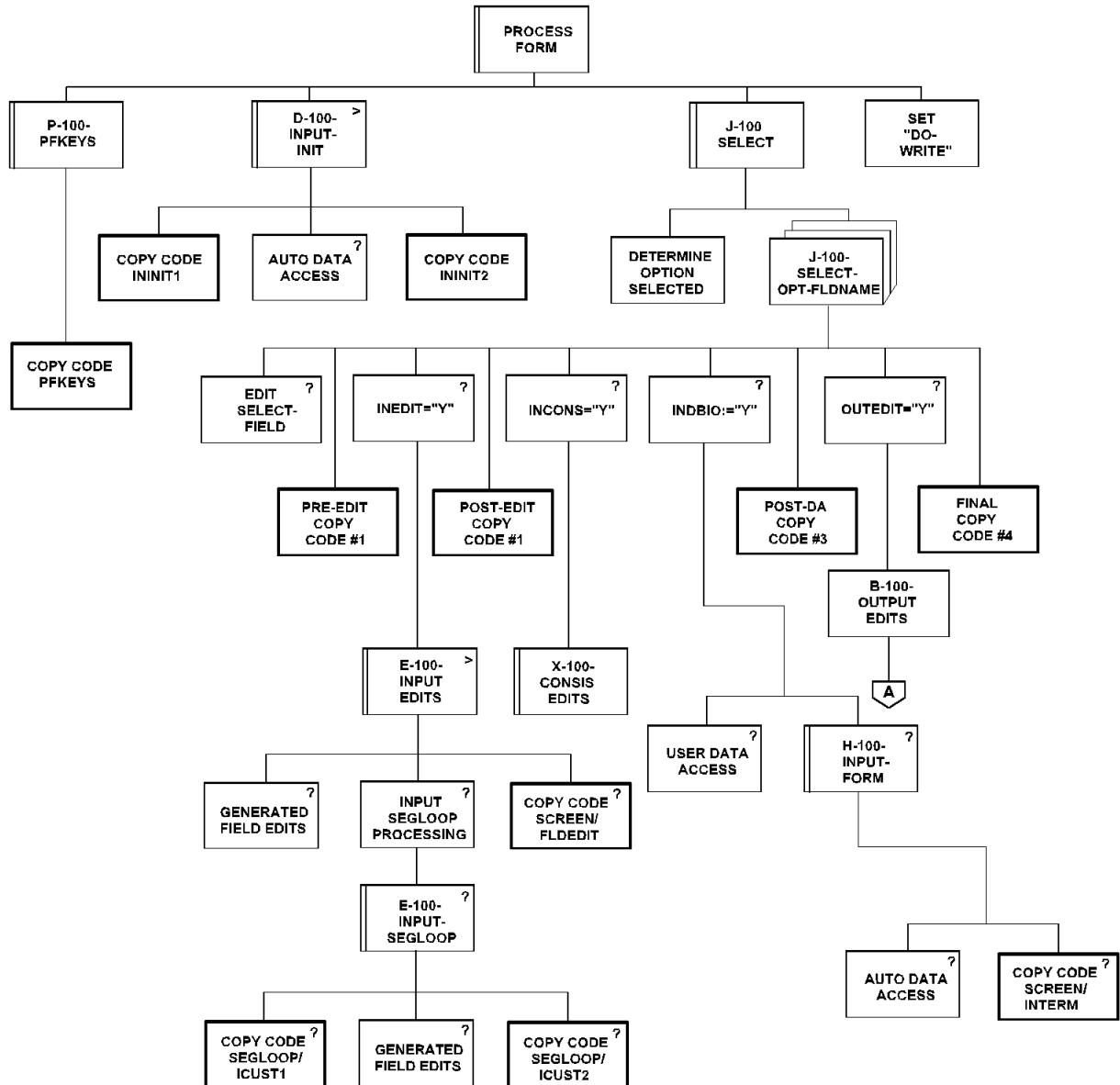
Finally, SET 'DO-WRITE' gains control and writes the screen to the terminal, thereby completing MAIN-OUTPUT.

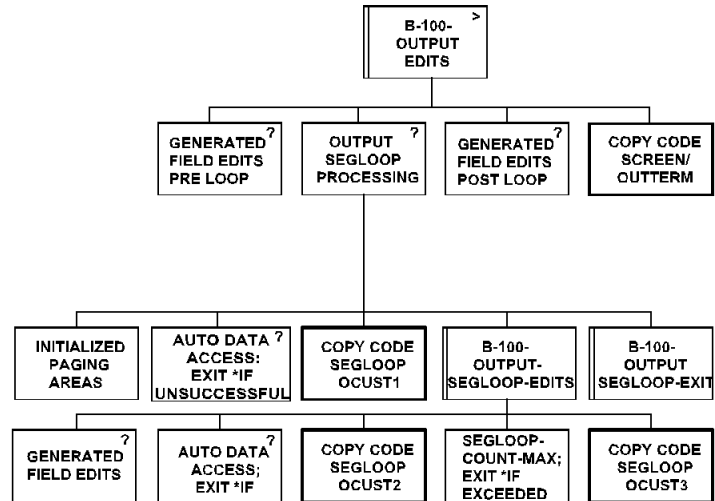
At this point, output processing is complete and input processing of the screen can begin.

PROCESS-FORM Processing

The PROCESS-FORM process, as shown in hierarchical chart for PROCESS-FORM processing, consists of P-100-PFKEYS, D-100-INPUT-INIT, J-100-SELECT, and a set DO-WRITE.

Note: If an error is detected during input processing, an error message is sent to the application user's terminal.

PROCESS-FORM process hierarchical chart

PROCESS-FORM process hierarchical chart (continued)**P-100-PFKEYS**

P-100-PFKEYS comprises the possibility of PF-key processing inherent to the screen (such as paging PF keys for list processing). If you define a SEGLOOP, CA Telon automatically creates this section in your program.

P-100-PFKEYS is followed by the COPY code requested by the PFKEYS parameter on the Create/Update Screen Definition screen.

D-100-INPUT-INIT

D-100-INPUT-INIT consists of the following:

- Possible Custom COPY Code requested by the ININIT1 parameter on the Create/Update Screen Definition screen
- AUTO DATA ACCESS requested using the Create/Update Data Group screen
- Custom COPY Code requested by the ININIT2 parameter on the Create/Update Screen Definition screen

D-100-INPUT-INIT is bypassable depending on processing performed in P-100-PFKEYS.

J-100-SELECT

J-100-SELECT consists of DETERMINE OPTION SELECTED, and J-100-SELECT OPT-fldname.

J-100-SELECT contains the following components:

- DETERMINE OPTION SELECTED
- J-100-SELECT-OPT-fldname , which exists if a non-blank value was entered for the SELECT FIELDS parameter on the Create/Update Screen Definition screen.

Following J-100-SELECT-OPT-fldname are EDIT-SELECT-FIELD, PERFORM E-100-INPUT-EDITS INEDIT logic, PERFORM X-100-CONSIS-EDITS INCONS logic, PERFORM H-100 or U-100 INDBIO logic, and PERFORM B-100-OUTPUT-EDITS OUTEDIT logic. Note that all of this logic is conditional. Additionally, note that there is the possibility of custom code points following each of the above PERFORMs.

- EDIT-SELECT-FIELD exists based on specifications entered on the Panel Definition menu.
- The Server pre-input edit custom code point is processed next.
- A PERFORM of E-100-INPUT-EDITS is processed next, if INEDIT=**Y** was specified on either the Update Select Field screen or the Update Select Parameters screen. E-100-INPUT-EDITS entails the following processing:
 - GENERATED FIELD EDITS exists only if you code the FLDTYPE field on the Update Panel Fields screen for an INPUT or OUTIN panel field.
 - INPUT SEGLOOP PROCESSING (SEGLOOP Mapping) exists only if there is an INPUT or OUTIN type SEGLOOP statement in the program definition. INPUT SEGLOOP PROCESSING incorporates E-100-INPUT-SEGLOOP which conditionally processes each input group, performing:

The COPY code associated with ICUST1 as specified for the ICUST1 field on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.

Any GENERATED FIELD EDITS that were coded using the FLDTYPE field on the Update Panel Fields screen for an INPUT or OUTIN panel field.

The COPY code associated with ICUST2 as specified for the ICUST2 field on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.

The COPY CODE associated with FLDEDIT exists only if you code the FLDEDIT field on the Create/Update Screen Definition screen.

- The Server post-input edit custom code point is processed next.
- A PERFORM of X-100-CONSIS-EDITS is processed next, if INCONS=**Y** was specified on either the Update Select Field screen or the Update Select Parameters screen. X-100-CONSIS-EDITS includes consistency edits either generated by the XEDIT, SEGEDIT, or SRC non-procedural statements, or COPY CODE requested by the CONSIS field.

Generated consistency edit code exists only if you specify, on the Panel Definition menu, that a XFEDIT, SEGEDIT, or SRC non-procedural statement has been specified under a non-SELECT type field. The COPY CODE exists only if you specify the CONSYS field on the Create/Update Screen Definition screen.

- The H-100 or U-100 INDBIO logic exists only if INDBIO=**Y** (for H-100) or INDBIO=**U** was specified on either the Update Select Field screen or the Update Select Parameters screen. If INDBIO=**Y**, the H-100-INPUT-TERM section is performed. H-100-INPUT-TERM incorporates the possibility of AUTO DATA ACCESS specified on the Create/Update Data Group screen and COPY CODE specified for the INTERM field on the Create/Update Screen Definition screen.

If INDBIO=**U**, the U-100 logic specified in Data Access with the same name as the SELECT field parameter on the Create/Update Screen Definition screen is performed.

- The Server post-data access custom code point is processed next.
- A PERFORM of B-100-OUTPUT-EDITS is processed next, if OUTEDIT=**Y** was specified on either the Update Select Field screen or the Update Select Parameters screen.

The B-100-OUTPUT-EDITS section handles editing and formatting of all OUTPUT/OUTIN fields. Additionally, it contains the code generated when an OUTPUT/OUTIN SEGLOOP is required.

The GENERATED FIELD EDITS *function* contains code specified by the FLDTYPE parameter on the Update Panel Fields screen (for example, to edit any OUTPUT or OUTIN fields).

OUTPUT SEGLOOP PROCESSING (SEGLOOP Mapping) exists only if there is an OUTPUT or OUTIN type SEGLOOP specified in the program definition. When SEGLOOP Mapping exists, it can include any of the following: INITIALIZED PAGING AREAS (Page Initialization), AUTO DATA ACCESS, a check for the Success or Omission of the Auto Calls , B-100-OUTPUT-SEGLOOP- EDITS, and B-100-OUTPUT-SEGLOOP-EXIT.

These are generated in the order specified. Note that:

- INITIALIZED PAGING AREAS exists only if PAGE=**Y** is specified on the Create/Update File SEGLOOP screen.
- AUTO DATA ACCESS exists if AUTOEXEC BROWSE was specified on the Create/Update Data Group screen. If the automatic access is unsuccessful, the loop is immediately exited to process post-loop generated field edits.

- During execution, B-100-OUTPUT-SEGLOOP-EDITS is repeated zero to n times, where n is the number of groups in the SEGLOOP (as determined by the INCRE parameter on the Create/Update Table SEGLOOP screen). When the repetition is complete, the box following the B-100-OUTPUT-SEGLOOP-EDITS gains control. (Specifically, this is SET 'DO-WRITE'.)

To process each group of fields in the loop, the program executes GENERATED FIELD EDITS through OCUST3. This logic includes the possibility of: GENERATED FIELD EDITS, AUTO DATA ACCESS , and OCUST2 COPY code. A check for maximum SEGLOOP-COUNT is executed next, followed by the possibility of OCUST3 COPY code.

GENERATED FIELD EDITS exists unless all of the OUTPUT or OUTIN fields specify FLDTYPE=**NONE** on the Update Panel Fields screen, Update Output, Input, Outin Fields screen, or Update Select Field screen. It is possible, but very unusual, that GENERATED FIELD EDITS will not exist.

AUTO DATA ACCESS exists whenever OCUST1 COPY code exists (for example, when AUTOEXEC is specified on the Create/Update Data Group screen). If the auto access is unsuccessful, the loop is immediately exited to process post-loop generated field edits.

SEGLOOP-COUNT-MAX checks to determine if the information retrieved from AUTO DATA ACCESS fits on the current screen. If not, AUTO DATA ACCESS is terminated; otherwise, if the loop continues back to GENERATED FIELD EDITS, OCUST3 COPY code receives control first.

Edits are performed for any fields defined after the loop.

- B-100-OUTPUT-SEGLOOP=EXIT exits from OUTPUT SEGLOOP PROCESSING.

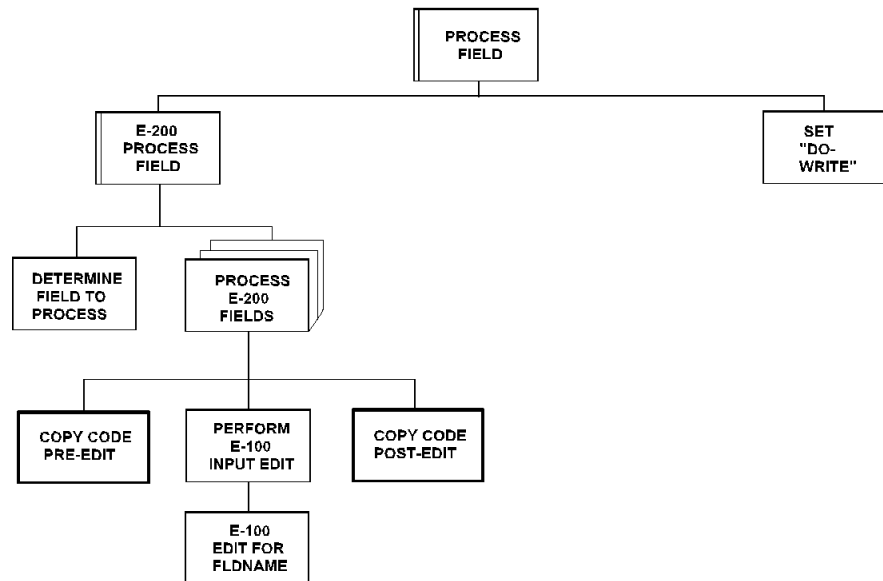
OUTTERM, if specified on the Create/Update Screen Definition, is processed next.

- The Server final custom code point is processed next.

PROCESS-FIELD Processing

The PROCESS-FIELD process, as shown in hierarchical chart, consists of E-200-PROCESS-FIELD, and a set DO-WRITE.

Note: If an error is detected during input processing, an error message is sent to the application user's terminal.

PROCESS-FIELD process hierarchical chart

E-200-PROCESS-FIELD

The E-200-PROCESS-FIELD generated logic first determines which field has been identified to be processed. When this has been accomplished, there are the following sequence of steps for the identified field:

- First, any specified pre-edit COPY CODE is processed.
- Next, the E-100-EDIT-fllname section is performed for the specified field.
- Finally, any specified post-edit COPY CODE is processed.

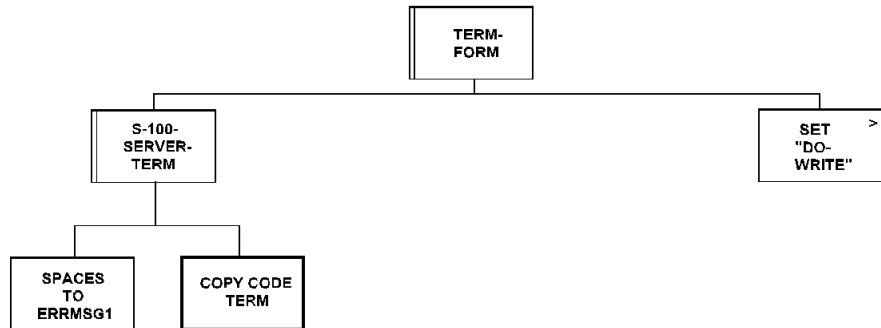
SET 'DO-WRITE'

SET 'DO-WRITE' causes processing to transfer control to the Client program.

TERM-FORM Processing

The TERM-FORM process, as shown in hierarchical chart, consists of S-100-SERVER-TERM, and a set DO-WRITE.

Note: If an error is detected during input processing, an error message is sent to the application user's terminal.

TERM-FORM process hierarchical chart**S-100-SERVER-TERM**

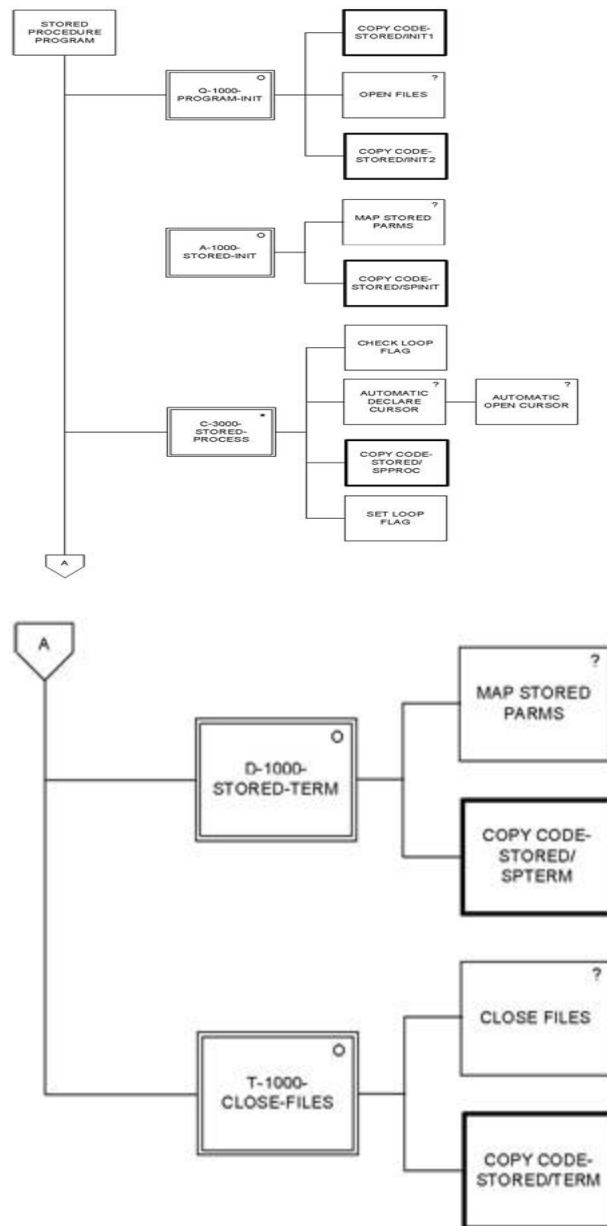
The S-100-SERVER-TERM generated logic first moves spaces to TPO-ERRMSG1. Then, any Server final COPY CODE is processed.

Stored Procedure Hierarchical Charts

This subsection discusses stored procedure program hierarchical charts in terms of a program overview and main processing.

Program Overview

The procedural code of a generated stored procedure program has a simple linear structure: initialization, processing, and termination. This program model does not make use of control indicators, as do other models. Following is the program flow diagram of a generated stored procedure program:



Open Files

The first section that is performed is Q-100-OPEN-FILES (similar name for PL/I). The Q-100 section generates the OPEN of any sequential or VSAM file access defined for the stored procedure. There are two custom code points associated with this section: INIT1, located prior to any file opens, and INIT2, located after file opens.

Initialize

The A-1000 section performs two functions: (1) it moves contents of IN and INOUT linkage parameters to their working-storage counterparts (if DBNAME has been defined for the parameter), and (2) optionally copies in the SPINIT custom code member, if specified.

Process

The program process section C-3000-STORED-PROCESS section is invoked as a oop (PERFORM UNTIL in COBOL). It contains generated code dealing with the termination of that loop, and also the possibility of AUTOEXEC data access, along with a SPPROC custom code point. A variable STP-LOOP-FLAG is initialized to LOW-VALUES at the beginning of the stored procedure, and is now tested at the beginning of the C-3000 section for SPACE. If that test is TRUE, then a jump is made to the end of the section, and the loop is discontinued. Next, any AUTOEXEC processing that has been specified is generated; this would occur only for an AUTOEXEC BROWSE or AUTOEXEC TRANSACT data access request for a DB2 table. Note that the code generated in such requests is abbreviated from that found in other program models: the DECLARE...CURSOR statement is generated, but the FETCH is not. The FETCH will be issued in programs which call this stored procedure: note the important coordination required in the definition of data access requests in the calling and called programs. Once the AUTOEXEC processing (if requested) is generated, then the SPPROC custom code point (if specified) is copied. This is followed by setting the STP-LOOP-FLAG to a value of SPACE. Note that by doing so, the loop will only be performed once, unless there is some logic in the SPPROC custom code member that conditionally sets the value of STP-LOOP-FLAG. Obviously you must be concerned not to allow the stored procedure to fall into an infinite loop, because of faulty logic coded relative to the STP-LOOP-FLAG.

Terminate

The D-1000 section performs two functions: (1) optionally copies in the SPTERM custom code member, if specified, and (2) moves contents of OUT and INOUT parameters to linkage parameters from their working-storage counterparts (if DBNAME has been defined for the parameter).

Close Files

The program concludes with the performance of the T-100-CLOSE-FILES section (similar names for PL/I). The T-100 section closes any sequential or VSAM files that were opened in the corresponding Q-100-OPEN-FILES section. Prior to the beginning of the close file logic, there is the TERM custom code point.

Other Sections

A stored procedure program will contain other sections such as U-100-USER-IO and Z-100-SECTION-COPY, if so instructed in the program definition. Additionally, there are the usual Z-900-SECTION-FALLOUT, Z-980-ABNORMAL-TERM, and Z-990-PROGRAM-ERROR sections generated at the end of the program.

Chapter 8: Custom Code

For purposes of this discussion, custom code is any COBOL or PL/I statements that add processing logic to the code generated automatically by CA Telon. While CA Telon automates virtually all routine programming, and in addition provides many capabilities beyond just simple automation, many applications have unique coding requirements. Custom code is used to incorporate these unique functional requirements into your CA Telon program.

This chapter first introduces the basics of inserting custom code in a screen definition. It then describes each keyword, in the TDF screens, used to define custom code to be included in your application, with reference to where the CA Telon Generator places the custom code in the generated program.

Basics of Using Custom Code

Although you can modify the CA Telon-generated program to change the logic, this makes it impossible to maintain the application system through CA Telon parameters, as is recommended. Therefore, to provide the capability of program modification but still allow high-level maintenance, CA Telon allows you to identify (in the high-level statements) custom code to be inserted at points throughout the generated program. Each such point is identified by a TDF parameter. Using these parameters, you name the block of code that contains the logic to be included in your application.

The code to be inserted can be stored either *in* the CA Telon Design Facility or as a member in a temporary PDS created from a copy library. When CA Telon generates the source code, it looks first in the program definition (SD, BD, and so on) for the COPY member referenced. If it cannot find it there, CA Telon looks in copy libraries designated in your JCL. In any case, CA Telon places the contents of the COPY member in the generated program, at a position predefined for the parameter. In the following example, the CONSID field on the Create/Update Screen Definition screen references the COPY member RMREQ. The member is shown in custom code (COBOL and PL/I) and is contained in CA Telon itself. By definition, this COPY member contains logic to be included in the consistency edits section of the program. CONSID logic is always placed in the X-100-CONSID-EDITS section.

Examples of generated source code show the part of the generated program that incorporates the RMREQ CONSID logic. Note that CA Telon also marks every set of custom code with a block of comments in the generated program so you can easily differentiate your custom code from the CA Telon-generated code.

For each parameter that can identify custom code, there is a specific purpose for the code referenced. There also is a specific point, in the generated program, where CA Telon places the COPY member code. For further information, see Chapter 7, "Program Hierarchical Structure."

Update Screen Definition

```

XXXXXX.SD UPDATE SCREEN DEFINITION ** *****
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV IMS _ SCRN PARMS _

GENERAL: DESC MVS CICS TARGET - COB VSAM ADD _____ REMARKS **DFLT**
*      NEXTPGM _____ CURSOR NAME _____ SIZE 24 X 80 LANG COB (COB/PLI)
*      CMPLOPT _____ IDENTIF _____ PROCEDR _____
DATA   XFERWKA TRXFERW _____
AREAS: _ WKAREA **DFLT**

OUTPUT:
A-100 _ OINIT1 OINIT1 _ OINIT2 **DFLT** _ CURSCUS _____
B-100 _ OUTTERM _____

INPUT:
P-100 _ PRKEYS ADD,ENT,1,2,3,4,5,6,OT _____
D-100 _ ININIT1 ININIT _ ININIT2 **DFLT**

E-100 _ FLDEDIT _____
X-100 + SCREEN XFEDIT/SEGEDIT _ CONSIS REMREQ
H-100 _ INTERM _____

MISC:  SECTION INITROOT
*      PGM CUST

```

Custom code (COBOL)

```

EDIT ---- XXXXXX.SD RMREQ MEMBER 001 OF 001 SIZE 000000 COL 07
COMMAND ==> SCROLL ==> CSR
***** ***** TOP OF DATA *****

35*****
36 * ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS *
37 *                                                                 *
38 IF NEXT-PROGRAM-NAME-ID = 'MV20' OR 'MV30'
39   IF XFER-ROOM-ID = SPACES OR ERROR-REQ-CHAR
40     MOVE ERROR-REQ-CHAR TO TPI-ID
41     MOVE 'E' TO CONSIS-INPUT-ERRORS
42     MOVE ERROR-ATTR TO TPO-ID-ATTR, TPO-OPTION-ATTR
43     MOVE '**ROOM ID REQUIRED FOR OPTION CHOSEN**' TO TPO-ERRMSG1
44     GO TO X-100-CONSIG-EDITS-RETURN.

***** ***** BOTTOM OF DATA *****

```

Custom code (PL/I)

```

EDIT ---- XXXXXX.SD RMREQ MEMBER 001 OF 001 SIZE 000000 COL 07
COMMAND ==> SCROLL ==> CSR
***** ***** TOP OF DATA *****

34 /***** */
35 /* ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS */
36 /* */
37 IF NEXT_PROGRAM_NAME_ID = 'PD20'
38     NEXT_PROGRAM_NAME_ID = 'PD30'
39 THEN IF XFER_ROOM_ID = ' '
40     XFER_ROOM_ID = ERROR_REQ_CHAR
41 THEN DO;
42     TPI_ID = ERROR_REQ_CHAR;
43     CONSIG_INPUT_ERRORS = 'E';
44     TPO_ID_ATTR = ERROR_ATTR;
45     TPO_OPTION_ATTR = ERROR_ATTR;
46     TPO_ERRMSG1 = 'ROOM ID REQUIRED FOR OPTION CHOSEN';
47     GOT0 X_100_CONSIG_EDITS_RETURN;
48 END;
***** ***** BOTTOM OF DATA *****

```

COBOL source code showing CONSIG logic

IDENTIFICATION DIVISION.

PROGRAM ID. MRCPMV10.

.

X-100-CONSIG-EDITS SECTION.

.

.

TELON-----

* COPY CONSIG

* ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS *

* *

IF NEXT-PROGRAM-NAME-ID = 'MV20' OR 'MV30'

.

.

X-100-CONSIG-EDITS-RETURN.

PL/I source code showing CONSYS logic

```
MRPPD10: PROC (DFHEIPTR,COMPRT) OPTIONS (MAIN REENTRANT);
.
X_100_CONSIS_EDITS: PROC;
.
/*TELON-----*/
/* %INCLUDE CONSYS */
/*-----*/
/***** */
/* ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS * */
/* */
IF NEXT_PROGRAM_NAME_ID = 'PD20'
NEXT_PROGRAM_NAME_ID = 'PD30'
.
```

Parameters to Request Custom Code

You can request custom code by using one or more of the CA Telon screen definition parameters. Before reviewing custom code, it is important that you understand the following terms:

- Position in program—The COBOL paragraph (or PL/I procedure) where CA Telon places the code in the program
- Contents of COPY member—The processing generally performed by the code referenced by the parameter (using the general format
PARAMETER=member-name)

Note: In addition to the parameters presented here, you can also customize CA Telon to include custom code automatically at the beginning or end of most CA Telon sections/procedures. The PGMCUSTOM field is used for this. PGMCUSTOM is on the Update Program Definition Defaults, Create/Update IMS/DC Driver Definition, and Create/Update Batch Definition screens.

Because of potential misuse, most installations restrict its use to an administrator or manager/coordinator. Check with your technical support group for applicability at your site.

Custom Code Point Name	Where Code Is Inserted	How Custom Code Point Is Used
CONSYS	X-100-CONSYS-EDITS	Logic to be performed after all consistency edits have been passed successfully. Generally used to perform cross-field and file checks, set indicators, etc. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver Definition, Create/Update IMS/DC Report Definition, or Create/Update Batch

Custom Code Point Name	Where Code Is Inserted	How Custom Code Point Is Used
		Definition screens.
CURSCUS	N-100-CURSOR-POSITION	Logic to handle cursor positioning. Applies when the cursor is not positioned at the same field each time the screen is displayed. You specify this parameter on the Create/Update Screen Definition screen.
ENDTRAN	C-1000-GET-TRAN	Logic to be performed after all transactions with a matching key have been fetched (batch match only). You specify this parameter on the Create/Update Batch Definition screen.
FLDEDIT	E-100-INPUT-EDITS	Logic to be performed after all field edits have been completed. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver Definition, Create/Update IMS/DC Report Definition, or Create/Update Batch Definition screens.
FMTCUST	B-5000-FORMAT-groupname B-N500-FORMAT-groupname	Logic to be performed at the end of the format section for the associated group.
GETTRAN	C-1000-GET-TRAN	Logic to be performed in the transaction-access section of the program, following any transaction automatic I/O. You specify this parameter on the Create/Update Batch Definition screen and the Create/Update Nonterminal Definition screen.
ICUST1	E-100-INPUT-EDITS	Logic to be performed for an input SEGLOOP prior to each iteration of input edits (within the loop). You specify this parameter on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.
ICUST2	E-100-INPUT-EDITS	Logic to be performed at the end of each iteration of input-screen loop processing. You specify this parameter on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screens. Note: ICUSTOM was used in Release 1.5. It is no longer used. ICUST2 replaces ICUSTOM.
IDENTIF	After IDENTIFICATION DIVISION statement	Valid for COBOL only. Specification of INITIAL and other options. You specify this parameter on the Create/Update Screen Definition (S110), Create/Update IMS Driver Definition (S210), Create/Update IMS Report Definition (S310), Create/Update CICS

Custom Code Point Name	Where Code Is Inserted	How Custom Code Point Is Used
		Nonterminal Definition (N110), and Create/Update Batch Definition (B110) screens.
INIT1	D-100-INPUT-INIT	Logic to be performed before the generated code that initializes fields and performs file I/O, and prior to the remainder of input processing. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver Definition, or Create/Update IMS/DC Report Definition.
INIT2	D-100-INPUT-INIT	Logic to be performed after the generated code that initializes fields and performs file I/O, and prior to the remainder of input processing. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver Definition, or Create/Update IMS/DC Report Definition. Note: INIT2 is a new name for earlier release INIT.
INIT1	Q-1000-INIT-PROGRAM Q-N100-INIT-PROGRAM	Logic to be performed after the optional generated code that parses input parms and before the optional generated code that opens files (for batch) or verifies a printer ID (for CICS Nonterminal). You specify this parameter on the Create/Update Batch Definition screen or the Create/Update Nonterminal Definition screen.
INIT2	Q-1000-INIT-PROGRAM Q-N100-INIT-PROGRAM	Logic to be performed after the optional generated code to open files (for batch) or verify a printer ID (for CICS Nonterminal) and before transaction handling. You specify this parameter on the Create/Update Batch Definition screen and the Create/Update Nonterminal Definition screen.
INMAST	C-1000-GET-MAST	Logic to be performed after each master record is retrieved and before matching logic is performed (batch match only). You specify this parameter on the Create/Update Batch Definition screen.
INTERM	H-100-INPUT-TERM	Logic to be performed at the end of input processing. You specify this parameter on the Create/Update Screen Definition screen.
INTRAN	C-1000-GET-TRAN	Logic to be performed after each transaction record is retrieved and before matching logic is performed (batch match only). You specify this parameter on the Create/Update Batch Definition

Custom Code Point Name	Where Code Is Inserted	How Custom Code Point Is Used
		screen.
MATCH	A-1000-MATCH	Logic to be performed after matching logic when master and transaction keys match (batch match only). You specify this parameter on the Create/Update Batch Definition screen.
MGREATER	A-1000-MATCH	Logic to be performed after matching logic when the master key is greater than the transaction key. You specify this parameter on the Create/Update Batch Definition screen.
OCUST1	B-100-OUTPUT-EDITS	Logic to be performed for output screen loop processing, after the first file access and before the first line is mapped to the screen. Generally equivalent to the combined OCUST2 and OCUST3 for subsequent iterations of the loop. You specify this parameter on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.
OCUST2	B-100-OUTPUT-SEGLOOP-EDITS	Logic to be performed during output-screen loop processing, after each subsequent file access. You specify this parameter on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.
OCUST3	B-100-OUTPUT-SEGLOOP-EDITS	Logic to be performed at the end of each iteration of output-screen loop processing, just before the iteration of the loop is displayed. You specify this parameter on the Create/Update Table SEGLOOP screen or the Create/Update File SEGLOOP screen.
OINIT1	A-100-OUTPUT-INIT	Logic to be performed before data files are accessed automatically to obtain output fields (for example, to perform special I/O, format fields not handled by FLDTYPE edits, initialize areas, or set indicators). You specify this parameter on the Create/Update Screen Definition screen or the Create/Update IMS/DC Report screen.
OINIT2	A-100-OUTPUT-INIT	Logic to be performed after data files are accessed automatically to obtain output fields. You specify this parameter on the Create/Update Screen Definition screen or the Create/Update IMS/DC Report screen.

Custom Code Point Name	Where Code Is Inserted	How Custom Code Point Is Used
OUTTERM	B-100-OUTPUT-EDITS	Logic to be performed at the end of output processing. You specify this parameter on the Create/Update Screen Definition screen or the Create/Update IMS/DC Report screen.
PFKEYS	P-100-PFKEYS	Logic to define the processing for each possible PF key selection from the screen (and, implicitly, which PF keys apply). While the code in this section is generally used for such PF key processing, it can be used for any other processing necessary at this point in the program. You specify this parameter on the Create/Update Screen Definition screen.
PRCTRAN	A-1000-PROCESS-TRAN A-N100-PR-TRAN	Logic to be performed in the transaction processing section, after the control switch has been set. You specify this parameter on the Create/Update Batch Definition screen and the Create/Update Nonterminal Definition screen.
PROCEDR	COBOL: After PROCEDURE DIVISION statement PL/I: in parentheses on Program PROC statement	Specifications of DECLARATIVES (for COBOL programs only) or processing Options (for PL/I only). You specify this parameter on the Create/Update Screen Definition (S110), Create/Update IMS Driver Definition (S210), Create/Update IMS Report Definition (S310), Create/Update CICS Nonterminal Definition (N110) and Create/Update Batch Definition (B110) screens.
REMARKS	REMARKS	The name of the custom code member added in the COBOL REMARKS section of the program, or at the beginning of the PL/I program. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver Definition, Create/Update IMS/DC Report Definition, or Create/Update Batch Definition screen.
SCONSIS	J-100-SELECT	Logic to be performed when one or more of the SELECT fields is specified (flow control processing, etc.). You specify this parameter on the Update Select Field screen or the Update Select Parameter screen.
SECTION	Extension of standard generated code	COBOL sections or PL/I procedures to be added to the program, and therefore made available for execution from other parts of the program. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver

Custom Code Point Name	Where Code Is Inserted	How Custom Code Point Is Used
		Definition, Create/Update IMS/DC Report Definition, or Create/Update Batch Definition screens.
TERM	T-100-PROGRAM-TERM T-N100-PROGRAM-TERM	The name of the custom code to be added in the program termination section of the program, prior to the closing of files in batch programs. You specify this parameter in the Create/Update Batch Definition or Create/Update Nonterminal Definition screen.
TGREATER	A-1000-MATCH	Logic to be performed after matching logic when the transaction key is greater than the match key (batch match only). You specify this parameter on the Create/Update Batch Definition screen.
WKAREA	Extension of variable storage	COBOL WORKING STORAGE areas or PL/I DECLARES required for this screen by the custom code included. You specify this parameter on the Create/Update Screen Definition, Create/Update IMS/DC Driver Definition, Create/Update IMS/DC Report Definition, Create/Update Batch Definition, or Create/Update Nonterminal Definition screen.
XFERWKA	Variable storage	The names of the custom code copy members that are to be added to the TRANSFER WORK AREA section of the program.

Custom Code Points for CICS Client

In the case of CICS Client/Server generation, the location of the custom code points in the client and server programs must be considered. The rule which is followed is to place all custom code points in the server program, with the following exceptions:

- Selected PFKEYS custom code members may be placed in the Client program by specifying these members in the CLIPFKS= parameter within TLNIIS.
- Selected SECTION custom code members may be placed in the Client program by specifying these members in the CLISECT= parameter within TLNIIS.
- WKAREA, XFERWKA, and REMARKS members are placed in both programs.
- Certain PGMCUST members will be placed in the client program, including C100, C200, C210, C220, C300, C500, C600, C700, C710, C730, C800, C810, C830, C930, C935, C940, C945, C948, K100, K200, L100, and M100.

Edits

CA Telon provides you with two types of edits for editing data: field edits and consistency edits. If you want to format a particular field or check it for valid format and/or content, you use a field edit. If you want to ensure that one or more fields are consistent with other data in the system, then you use a consistency edit. For example, if you want to compare two dates to ensure that one date is less than the other, you then use a consistency edit. A description of field edits follows.

Field Edits

Several screen definition parameters are used to indicate the type(s) of edits to be performed on the field, either to format the field or to verify its content. The edits are as follows:

- The **FLDTYPE** parameter defines the type of the panel field (date, dollar, state code, etc.) and dictates the corresponding type of edit logic to be included for the field by CA Telon.
- The **IEXTEND** and **OEXTEND** parameters allow additional variables to be passed to input and output field edits. CA Telon-supplied edits do not use these extensions. However, these parameters are provided for installation-supplied edits.
- The **FORMAT** parameter defines the picture format of the field, and dictates how CA Telon formats the field for output or input (generally by inserting or stripping special format characters, such as a hyphen in an otherwise numeric phone number).
- The **CONVERT** parameter defines values to which a field is converted. It dictates the contents of an associated conversion table. For input processing, it also requests logic to verify that the entered value is one for which a conversion entry has been defined.
- The **VALUES** parameter applies for input only and defines one or more acceptable values for the field. It dictates logic to verify the actual value at run time against those defined as acceptable.
- The **RANGE** parameter applies for input only and defines one or more acceptable ranges of numeric values for the field, and dictates logic to verify that an acceptable value is used at run time.

- The REQ parameter applies for input only and defines whether the field is required on screen entry.
- The PIC parameter applies for output only and overrides the default formatting characteristics of a FLDTYPE=**NUMERIC** field. It provides for generation of an appropriate picture format for the field in the output buffer (generally numeric editing, such as \$ZZ,ZZ9.99).
- The MAPOUT parameter applies to output only and allows for variable mapping to the output buffer. Through the use of custom code, you can control when a field is and is not mapped to the output screen.

Description of generated logic

Field edits are used during the mapping of data fields, either on output (when fields are moved from a file or work area to the screen) or on input (when fields are moved from the screen to a file or work area). On output, a field edit, at most, reformats the field as it is moved. On input, a field edit first checks the field for valid format and/or content before reformatting it. Notice, therefore, that an input edit can return an error condition to the program, whereas this does not apply for output edits.

Edit logic for each field is generated into program sections B-100-OUTPUT-EDITS and E-100-INPUT-EDITS. The usage specification—OUTPUT, INPUT, or OUTIN—determines whether the edit is an output and/or input edit.

Field edit parameters

The following discussion provides more information about each of the types of edit specifications, in order as listed above. For further details about the specifications, refer to the *Design Facility Reference*.

Note: FLDTYPE, CONVERT, FORMAT, and VALUES are mutually exclusive; for each field, you can specify only one of these parameters.

For complete information on the FLDTYPE parameters, refer to the *Design Facility Reference*.

FLDTYPE parameter

The FLDTYPE parameter defines the screen field's data type (date, dollar, state code, etc.) and it dictates the corresponding type of edit logic to be included for the field by CA Telon. It is the main edit parameter in the screen definition, and it controls the formatting of data as well as edit procedures.

For OUTPUT and INPUT fields, you specify one FLDTYPE parameter. For OUTIN fields, you can specify two parameters: the first for output and the second for input use. (If you specify only one parameter for an OUTIN field, the corresponding edit applies both on output and input).

The format of the FLDTYPE parameter follows:

FLDTYPE=edit-type (for single-edit specifications)

or

FLDTYPE=(output-edit,input-edit) (for OUTIN double edit)

CA Telon provides several standard FLDTYPE edits. In addition, you can include a keyword requesting a user-supplied subroutine as the FLDTYPE specification to instruct CA Telon to execute that subroutine to edit the field. For instructions to code user-supplied routines, refer to the *Design Facility Reference*.

Each edit routine processes either numeric or alphanumeric data. Numeric data routines are limited to only those items defined as numeric in your files or processing work area. When specifying a routine for a field, make sure you select the appropriate routine by data type. Each routine is identified either as numeric or alphanumeric. For identification on each routine, refer to the *Design Facility Reference*.

CA Telon provides you with two FLDTYPE parameter values for describing special non-edits as follows:

- NONE—No field-formatting or editing logic is automatically generated by CA Telon; however, you can code such logic and insert it into your program using custom code. NONE is the default when no DBNAME has been defined for the field.
- ALPHA—The field is moved from or to the screen as is, with no editing or formatting. This is the default when a DBNAME has been defined for the field.

FORMAT parameter

The FORMAT parameter dictates a special way for CA Telon to format an alphanumeric field for output or input (generally by inserting or stripping special format characters, such as a hyphen in an otherwise numeric phone number).

The syntax of the FORMAT parameter follows:

FORMAT='mask'

Specify the mask using Xs to indicate characters to be moved as is; 9s to be verified as numeric on input but otherwise to be moved as is; and any other characters to be inserted in the corresponding position(s) in the field on output and stripped from the corresponding position(s) on input.

Specify the same length for the mask as you specified for the field as displayed on the screen. Make the length of the target field, however, only as long as the number of 9s and Xs in the mask, combined. If the target field is longer than the total number of 9s and Xs, CA Telon blank-pads it to the right during mapping. Remember that the target field must be alphanumeric, even for fields whose map comprises all 9s. In the following specification, FLD3 represents a social security number:

```
FLD3 FIELD OUTIN,POS=(12,12),
      FORMAT='999-99-9999'
```

On output, CA Telon moves the number from a file or work-area field to the screen, formatting it with hyphens as shown in the mask. On input, it first verifies that the characters specified as 9s are entered as numeric digits and then stores those characters (for example, strips the hyphens) in the target alphanumeric file or work area.

In the example below, which illustrates X format characters as well as the 9s, CA Telon proceeds as described above except that, on input, it does a straight move of those characters without verifying the contents of the X characters. Notice that the formatting characters @@@ are included as well as a hyphen. These characters are inserted during output and stripped during input, just as the hyphen characters illustrated above (for example, ABC-10-@@@Z).

```
FLD4 FIELD OUTIN,POS=(12,12),
      FORMAT='XXX-99-@@@X'
```

The FORMAT specification is handled internally by CA Telon in supplied subroutines.

CONVERT parameter

The CONVERT parameter defines values to which a field is converted and, implicitly, the acceptable values for that field. Specifically, CONVERT defines one or more pairs of values, the first of which defines the field's value as it appears on the screen, the second of which defines the field's value as it is stored.

The format of the CONVERT parameter follows:

```
CONVERT=(screen-val1,stored-val1
        [,screen-val2,stored-val2]..)
```

The CONVERT parameter generates a table of values, which CA Telon searches at run time. If you have specified more than one pair of values, all the first values in the pairs must be of equal length; similarly, all the second values in the pairs must be of equal length. If the significant digits/characters in one of the pairs is shorter than those in another pair, you must blank-pad or zero-fill the shorter specification. Note that surrounding quotes are necessary for those values that contain special characters or spaces; otherwise—including the case where you zero-fill a value—you need not use quotation marks.

For example, the following output field specification converts a code for display on the screen, changing an M code to MALE and an F code to FEMALE:

```
FLD1 FIELD OUTPUT,POS=(12,12),  
      CONVERT=('MALE ',M,FEMALE,F)
```

On output, CA Telon searches the conversion table for a match with the second value in a pair and converts that value to the corresponding first value in the pair. If there is no match, the output field is left at its initial value.

On input, CA Telon reverses this process to search for a match with the first value in a pair and convert it to the corresponding second value. If the input value does not match one of the table entries, CA Telon flags an error.

Another example of an OUTIN field follows. In this example, CA Telon converts A, B, and C to A11A, B22B, and C33C, respectively, on input. (If a value other than A, B, or C is entered, CA Telon returns an error message.) On output, CA Telon converts the stored values A11A, B22B, and C33C to A, B, and C, respectively. If it encounters a stored value other than A11A, B22B, or C33C, CA Telon moves spaces to the screen.

```
FLD2 FIELD OUTIN,POS=(10,35),  
      CONVERT=(A,A11A,B,B22B,C,C33C)
```

If one of the conversion values in the above example had less than four digits, you then specify it either in quotation marks with a blank or zero-filled, as follows:

```
FLD2 FIELD OUTIN,POS=(10,35),  
      CONVERT=(A,A11A,B, ' 22B',C,C33C)
```

VALUES parameter

The VALUES parameter applies for input only. It defines one or more acceptable values for the field. The format of the VALUES parameter follows:

```
VALUES=(value1 [,value2]...)
```

Using the values defined, CA Telon generates 88-level items (for COBOL) or a search array (for PL/I). At run time, if any value other than the one defined is entered, CA Telon detects an error.

To specify the values, include them just as you want them to appear on the screen. Surrounding quotes are necessary for those values that contain special characters or spaces; otherwise, quotes are not required (but are allowed).

CA Telon recommends that all values be the same length and agree with the LTH field specification. However, this is not required. If a value as specified is longer than the field length (LTH), then at compile time, COBOL gives you a diagnostic and PL/I truncates the value to the LTH field specification.

In the following input field specification, only VALUES 10, 12, and AL are acceptable on input:

```
FLD1 FIELD INPUT,POS=(12,12),LTH=02,
      VALUES=(10,12,AL)
```

RANGE parameter

The RANGE parameter defines one or more acceptable ranges of values for a numeric field. It applies only to numeric type fields; that is, only to fields whose FLDTYPE specification references a numeric edit (FLDTYPE=**NUMERIC**, FLDTYPE=**FLOAT**, etc.). The range is inclusive; that is, the numbers defining the bounds are considered valid. The format of the RANGE parameter follows:

```
RANGE=(start-range1,end-range1
      [,start-range2,end-range2]...)
```

In the following example, the field is considered valid if, on input, it has a value between 4 and 9, inclusive:

```
FLD5 FIELD INPUT,POS=(22,02),FLDTYPE=NUMERIC,
      RANGE=(4,9)
```

REQ parameter

The REQ parameter applies to INPUT and OUTIN fields only, and defines whether the field is required on screen entry. By default, REQ=**N** (no) for any field.

If you specify REQ=**Y** (yes), CA Telon ensures that a non-blank value was entered for the field. If a value was not entered, CA Telon returns an application-defined character to the screen (defined by ERROR-REQ-CHAR in the hhWKAREA COPY member) and indicates an error. For an OUTIN field, REQ=**Y** does not require that the operator key in the field, but does require that the field not be spaces when Enter is pressed.

If you specify that REQ=**C** (consistency), CA Telon does not check whether a non-blank value was entered for the field, but leaves checking to consistency code that can be specified either using custom code or by using the SEGEDIT or XFEDIT statements. (In a consistency edit, you can specify conditions under which a field is required.) The format of the REQ parameter follows:

Parameter	Description
REQ= Y	To define the field as required
REQ= N	To define the field as not required
REQ= C	To define that the field is not required by CA Telon-generated logic, but can be required by conditions specified by consistency code

PIC parameter

The PIC parameter applies for OUTPUT or OUTIN fields whose FLDTYPE is specified as **NUMERIC**, **FULLNUM**, or **DOLLAR**. It overrides the standard output formatting for those fields. This standard formatting is shown below for a five-digit number:

FLDTYPE Specification	span=rest COBOL PICTURE	DEFAULT PL/I
	COBOL	PL/I
NUMERIC	Z(4)9.	(4)Z9
FULLNUM	9(5)	(5)9
DOLLAR	Z(2)9.99	(2)Z.9.99

Using the PIC specification, CA Telon generates an appropriate picture format for the field in the output buffer. You define the PIC in terms of any standard COBOL or PL/I PICTURE clause, following the syntax below:

PIC='picture-clause'

On input, the PIC parameter is ignored.

MAPOUT parameter

The MAPOUT parameter allows variable mapping of a field to the output buffer. The MAPOUT parameter indicates the name of the COBOL or PL/I data field whose value controls whether mapping of the field takes place. When the value of the indicated field is **Y**, the field is mapped on output. When its value is anything other than **Y**, the field is not mapped on output.

Consistency Edits

Consistency edits comprise processing logic to ensure that one or more fields are consistent with other data in the system, whether or not that data is input via the screen. For example, you might want a consistency edit to verify the uniqueness of a value just entered, or to verify two dates against each other to make sure one is less than the other.

As with other areas of CA Telon, consistency edits can be defined either through TDF parameters or through COBOL or PL/I procedural statements. The TDF parameters are preferred, of course, but sometimes they do not have the flexibility required for some editing. The two types of consistency edits are defined separately below. The same examples are used in both cases for comparative purposes.

CA Telon generated edit structure

The two TDF parameters that define consistency edits are XFEDIT and SEGEDIT. You use XFEDIT for the cross field editing of two or more fields. You use SEGEDIT to verify whether a segment or record exists based on a key value. The rest of this subsection explains, for both SEGEDIT and XFEDIT, the structure of the edit and how you insert custom statements using the SRC statement. A separate discussion of each of these parameters follows.

TDF edit structure

The structure of the edit for XFEDIT and SEGEDIT is the same, as defined below. If you require a different structure, then code the edit using procedural statements as defined later in this subsection. The standard edit structure always includes:

1. The check for the error condition—This might simply be the comparison of two or more fields, or it might include the read for a record, segment, or row.

The indication of an error condition by setting the error indicator CONSYS-INPUT-ERRORS (for mainline option 1) or CONTROL-INDICATOR (for mainline options 2 and 3).

2. The movement of an error message to the standard error field ERRMSG1. Note that you can either supply the error message within single quotes (') or the name of a field containing the error message.
3. The highlighting of one or more fields on the screen and the positioning of the cursor. Note that if the CURSOR parameter is used, the cursor is positioned at that field regardless of what other fields are highlighted. If the CURSOR parameter is not used, the cursor is positioned at the first highlighted field.
4. The exit to the end of the consistency-edit section.

The consistency edits from TDF parameters always appear as the first code in the consistency edit sections. Thus code inserted using the CONSYS or SCONSYS parameters are always inserted after the code generated from XFEDIT and SEGEDIT.

CA Telon generates the XFEDIT and SEGEDIT code in the X-100- CONSYS-EDITS section of a program except when an XFEDIT or SEGEDIT is defined for a SELECT field. These edits are defined from the Panel Definition Menu screen. In this case, CA Telon generates the edit code in J-100-SELECT. The edits will be generated following the other code for that SELECT field.

As an example, assume that three SELECT fields have been defined for a screen definition. The second SELECT field has an XFEDIT defined. When CA Telon generates the program, the code for that XFEDIT will appear in the J-100-SELECT paragraph, following any code generated for the second SELECT field.

The following illustrates this point:

J-100-SELECT-xxx

1st select statement generated code

.

.

2nd select statement generated code

.

generated XFEDIT code for 2nd select statement

.

.

3rd select statement code

SRC statement

The intent of TDF consistency edits is that they be executed in order as they appear in the screen definition, without regard to procedural logic in COBOL or PL/I. Sometimes, however, you can enhance the flexibility of those edits by inserting COBOL or PL/I statements between edits. You can do this using the SRC statement, which simply allows you to define the line to be inserted.

In the following example, the perform of a read is inserted between two non-procedural edits:

```
XXXXXX.PD LIST SRC, XFEDIT, SEGEDIT *****  
COMMAND ==> _____ PAGE 01
```

```
SEQ TYPE DESCRIPTION (FIRST WORD IS XFEDIT/SEGEDIT NAME) OR STATEMENT CODE
```

```
___ SRC Perform U-100-READ-TRGEMPL
```

```
___ _____  
___ _____
```

Note: COBOL IF statements always terminate with the last line of code generated by the consistency edit. That is, for COBOL code generated by a consistency edit, CA Telon places a period at the end of the last generated line.

If the edits are to be contained in a loop that processes a SEGLOOP on input, SRC statements are necessary. You must be sure to set up the loop and increment the subscript, SEGLOOP-COUNT. The following examples display this setup information.

For COBOL:

```

XXXXXX.PD LIST SRC, XFEDIT, SEGEDIT *****
COMMAND ==> _____ PAGE 01

SEQ TYPE DESCRIPTION (FIRST WORD IS XFEDIT/SEGEDIT NAME) OR STATEMENT CODE

---- SRC MOVE 1 TO SEGLOOP-COUNT.
---- SRC CONSIDER-EDIT-LOOP.
---- SEGEDIT MAKE SURE THAT EMPLOYEE EXISTS
---- SRC ADD 1 TO SEGLOOP-COUNT.
---- SRC IF SEGLOOP-COUNT < 9
---- SRC GO TO CONSIDER-EDIT-LOOP.

```

For PL/I:

```

XXXXXX.PD LIST SRC, XFEDIT, SEGEDIT *****
COMMAND ==> _____ PAGE 01

SEQ TYPE DESCRIPTION (FIRST WORD IS XFEDIT/SEGEDIT NAME) OR STATEMENT CODE

---- SRC DO SEGLOOP COUNT = 1 TO 8;
---- SEGEDIT MAKE SURE THAT EMPLOYEE EXISTS
---- SRC END;

```

Usage of XFEDIT

The XFEDIT is used to compare two or more fields to determine an error condition. Those fields can be in any area in working storage (including the input buffer, the transfer work area, and I/O buffers). The error condition is defined using the COND parameter and includes field names, literal values, or expressions separated by mnemonics (that is, LT, LE, EQ, GT). The THENIF mnemonic allows the nesting of conditions (that is, nesting of generated IFs). For more information, refer to the *Design Facility Reference*.

Use the HIGHLIGHT and ERRMSG parameters to define which fields on the screen are to be highlighted and the error message to be returned. Use the ERRCHAR parameter when the consistency edit is to simulate required field processing (the ERROR-REQ-CHAR is to be returned to some field or fields on the screen). If the CURSOR parameter is used, the cursor is positioned at that field regardless of which fields are highlighted. If the CURSOR parameter is not used, the cursor is positioned at the first highlighted field.

If you want to perform an XDFEDIT in a loop, to process for all iterations of a field (or fields) in a SEGLOOP, place a 'Y' in the SEGLOOP field.

Note: In order for the code to be generated correctly, you must specify an array subscript for each TPI-field (or other array fields) specified in a SEGLOOP XFEDIT. The best subscript to use for the TPI-fields is SEGLOOP-COUNT.

On the XFEDIT screens, commas or spaces can be used as separators.

```
XXXXXX.PD UPDATE XFEDIT *****  
COMMAND ==> _____  
-  
EDIT NAME XXXXXXXX  
  
COPY EDIT BASE: _____ SEGLOOP: __  
EDIT CONDITION: 'IN-DATE,LT,WORK-CURRENT-DATE'  
_____  
_____  
_____  
  
ERROR MESSAGE: 'DATE EXCEEDS TODAYS DATE'  
_____  
  
HIGHLIGHT FIELDS: INDT,FUNC  
_____  
  
ERRCHAR FIELDS: _____  
  
CURSOR AT FIELD: _____
```



```

XXXXXX.PD UPDATE XFEDIT *****
*
COMMAND ==> _____
-
EDIT NAME XXXXXXXX

COPY EDIT BASE: _____ SELOOP: ____
EDIT CONDITION: TPI-FUNC,EQ,"A",
                THENIF,IN-DATE,LT,WORK-CURRENT-DATE
                _____
                _____
                _____

ERROR MESSAGE: DATE EXCEEDS TODAYS DATE
                _____

HIGHLIGHT FIELDS: INDT,FUNC
                _____

ERRCHAR FIELDS: _____

CURSOR AT FIELD: _____

```

Usage of SEGEDIT

The SEGEDIT is used to determine whether a particular record or segment exists on a file or database. Usually this involves a key field entered on the screen. For example, in the case of an add function, if the key already exists on the file, then the SEGEDIT generates an error message. In the case of a display function, this generates an error if the key does not exist on the file.

The first parameter (positional) in SEGEDIT identifies the SEGMENT (by segment name) for the record or segment to be read. The SEGKEY parameter identifies the field containing the key for the record or segment to be read. If the SEGKEY parameter is not used on SEGEDIT, then CA Telon uses the SEGKEY of the SEGMENT for the read.

The ERROR parameter defines whether an error occurs when the data is found (ERROR=**FOUND**) or not found (ERROR=**NOTFOUND**, which is the default). Use the HILIGHT and ERRMSG parameters to define which fields on the screen are to be highlighted and the error message to be returned.

If you use the CURSOR parameter, the CURSOR is positioned at that field you specify regardless of which fields are highlighted. Otherwise, the cursor is positioned at the first highlighted field.

The WHEN parameter allows this edit to be conditionally executed (for example, only when an add function is requested). You can use other parameters to customize the read (for example, the DL/I function performed, the I/O area read into). An example of SEGEDIT:

```
XXXXXX.PD UPDATE SEGEDIT *****
COMMAND ==>
EDIT NAME CLAIMAIN
COPY EDIT BASE: _____ SEGLOOP: ____
SEGMENT NAME: CLAIM PCBNAME: _____
SEGKEY: XFER-CLAIM-NO_____
WHEN: _____
_____
_____

ERROR CONDITION: FOUND____
HIGHLIGHT FIELDS:CLMNO_____
ERRCHAR FIELDS: _____
CURSOR FIELD: _____
ERROR MESSAGE: CLAIM NO ALREADY EXISTS

CALL FUNC: _____ OPCODE: _____

DLI QUALIFY ____ CMCODE: _____ I/O AREA: _____
SSALIST _____
_____

VSAM SEGMENT LTH: _____
GEN KEY LTH: _____
```

If you want to perform a SEGEDIT in a loop, to process for all iterations of a field (or fields) in a SEGLOOP, place a 'Y' in the SEGLOOP field.

Note: In order for the code to be generated correctly, any SEGLOOP or array fields specified for the SEGEDIT must have a subscript. The best subscript to use for TPI-fields is SEGLOOP-COUNT.

Custom coded consistency edits

You can define procedural consistency checks in either of these two ways:

- Using the CONSIG field on the Create/Update Screen Definition screen, you can place consistency edits in the X-100-CONSIG-EDITS section of generated code.
- Using the SCONSIG field on the Create/Update Screen Definition screen, you can include consistency-edit logic for fields defined as SELECT. Such consistency-edit logic might, for example, verify the contents of data entered for the select field in conjunction with other existing data. SCONSIG edits fall in the J-100-SELECT section of the logic. (SCONSIG selection processing is discussed earlier in this chapter.)

Either the CONSYS or SCONSYS logic can be executed for the same screen, but not both. If there are no SELECT fields, then the CONSYS edits follow the E-100-INPUT-EDITS for the fields, assuming there are no edit errors. If there are SELECT fields, the SCONSYS edits follow the processing for the selection option and, if INEDIT=Y, the field edits.

Considerations

In putting together the consistency code to be included using CONSYS or SCONSYS, the following general design considerations are worth noting:

- For each consistency edit that fails, the message returned to the terminal must be explicit. Unlike the field edits, where a general purpose message suffices, problems raised during consistency edits are not always obvious.
- Because the message returned for each different consistency edit is unique, separate the logic so that only one edit is determined—and corrected—at a time. This requires a hierarchical approach to the order in which the edits are done (the most critical first, followed by the next most critical, etc.).
- All fields involved in the consistency edit must be highlighted. For example, if two values are not correct in relationship to each other (for example, a range is supplied incorrectly from high-to-low), the logic must highlight both values when it returns the error message.

If an error is detected in consistency-edit logic, the following processing must be followed:

1. Move an appropriate error message to the reserved field, TPO-ERRMSG1 (or to another appropriate output field defined on the screen).
2. Indicate an error condition by setting the error indicator CONSYS-INPUT-ERRORS (for mainline option 1) or CONTROL-INDICATOR (for mainline options 2 and 3).
3. Highlight the fields involved in the error, by moving the reserved field, ERROR-ATTR, to the attribute byte for those fields. When the screen is displayed, the cursor is positioned at the first highlighted field.
4. Skip to the end of the consistency-edit logic (X-100-CONSYS-EDITS-RETURN for CONSYS and J-100-SELECT-RETURN for SCONSYS).

The following code illustrates this processing.

- For COBOL with PGMSTRUCT 1:


```
//edit logic//
IF ...
    MOVE 'message' TO TPO-ERRMSG1
    MOVE 'E' TO CONSYS-INPUT-ERRORS
    MOVE ERROR-ATTR TO TPO-fieldname-ATTR
    GO TO X-100-CONSYS-EDITS-RETURN.
```

- For COBOL with PGMSTRUCT 2 or 3:

```
//edit logic//
IF ...
    MOVE 'message' TO TPO-ERRMSG1
    MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
    MOVE ERROR-ATTR TO TPO-fieldname-ATTR
GO TO X-100-CONSIS-EDITS-RETURN.
```

- For PL/I with PGMSTRUCT 1:

```
//edit logic//
IF... THEN DO;
    TPO_ERRMSG1 = 'message';
    CONSYS_INPUT_ERRORS = 'E';
    TPO_fieldname_ATTR = ERROR_ATTR;
    GO TO X_100_CONSYS_EDITS_RETURN;
END;
```

- For PL/I with PGMSTRUCT 2 or 3:

```
//edit logic//
IF... THEN DO;
    TPO_ERRMSG1 = 'message';
    CONTROL_INDICATOR = DO_WRITE_LIT;
    TPO_fieldname_ATTR = ERROR_ATTR;
    GO TO X_100_CONSYS_EDITS_RETURN;
END;
```

Examples of Consistency Checks

Two examples of procedural consistency edit code follow. The first example is a field-to-field consistency check. The second example is a field-to-database consistency check. Either one can appear in CONSYS or SCONSYS logic. For purposes of illustration, assume that the (first) field-to-field example is part of a CONSYS COPY code and that the database example is part of an SCONSYS code.

Field-to-field consistency check

The following example of consistency code parallels that shown above, and might be found in the CONSYS COPY member for a program. It checks an input date against the current date to make sure that it is not greater than the current date. If it is greater, it returns an error message and highlights the date fields. In this example, INDT is the name of the date field on the screen; IN-DATE is the name of the date field, in storage, to which the screen date field is mapped.

- For COBOL with PGMSTRUCT 1:
IF IN-DATE LESS THAN WORK-CURRENT-DATE
MOVE 'DATE EXCEEDS TODAY' 'S DATE' TO TPO-ERRMSG1
MOVE 'E' TO CONSIG-INPUT-ERRORS
MOVE ERROR-ATTR TO TPO-INDT-ATTR
GO TO X-100-CONSIG-EDITS-RETURN.
- For COBOL with PGMSTRUCT 2 or 3:
IF IN-DATE LESS THAN WORK-CURRENT-DATE
MOVE 'DATE EXCEEDS TODAY' 'S DATE' TO TPO-ERRMSG1
MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
MOVE ERROR-ATTR TO TPO-INDT-ATTR
GO TO X-100-CONSIG-EDITS-RETURN.
- For PL/I with PGMSTRUCT 1:
IF IN_DATE < WORK_CURRENT_DATE
THEN DO;
TPO_ERRMSG1 = 'DATE EXCEEDS TODAY' 'S DATE';
CONSIG_INPUT_ERRORS = 'E';
TPO_INDT_ATTR = ERROR_ATTR;
GO TO X_100_CONSIG_EDITS_RETURN;
END;
ELSE:
- For PL/I with PGMSTRUCT 2 or 3:
IF IN_DATE < WORK_CURRENT_DATE
THEN DO;
TPO_ERRMSG1 = 'DATE EXCEEDS TODAY' 'S DATE';
CONTROL_INDICATOR = DO_WRITE_LIT;
TPO_INDT_ATTR = ERROR_ATTR;
GO TO X_100_CONSIG_EDITS_RETURN;
END;
ELSE:

Field-to-database consistency check

The code below illustrates an SCONSID check. It verifies that, for a selected (new) claim number, a DL/I segment does not already exist. If it does exist already, the code returns an error message and highlights the claim number of the screen. In this example, CLMNO is the name of the claim-number field on the screen; XFER-CLAIM-NO is the name of the claim-number field, in storage, to which CLMNO is mapped.

- For COBOL with PGMSTRUCT 1:

```
MOVE XFER-CLAIM-NO TO CLAIMANT-SSA-VALUE.  
CALL 'CBLTDLI' USING GU-FUNC  
                        CLAIM-PCB  
                        IOA-CLAIMANT-SEGMENT  
                        CLAIMANT-SSA.  
IF CLAIM-STATUS-CODE = SPACES  
    MOVE 'CLAIM NO ALREADY EXISTS' TO TP0-ERRMSG1  
    MOVE 'E' TO CONSID-INPUT-ERRORS  
    MOVE ERROR-ATTR TO TP0-CLMNO-ATTR  
    GO TO J-100-SELECT-RETURN.  
ELSE . . .
```

- For COBOL with PGMSTRUCT 2 or 3:

```
MOVE XFER-CLAIM-NO TO CLAIMANT-SSA-VALUE.  
CALL 'CBLTDLI' USING GU-FUNC  
                        CLAIM-PCB  
                        IOA-CLAIMANT-SEGMENT  
                        CLAIMANT-SSA.  
IF CLAIM-STATUS-CODE = SPACES  
    MOVE 'CLAIM NO ALREADY EXISTS' TO TP0-ERRMSG1  
    MOVE DO-WRITE-LIT TO CONTROL-INDICATOR  
    MOVE ERROR-ATTR TO TP0-CLMNO-ATTR  
    GO TO J-100-SELECT-RETURN.  
ELSE . . .
```

- For PL/I with PGMSTRUCT 1:


```
CLAIMANT_SSA_VALUE= XFER_CLAIM_NO;
DLI_PARM_COUNT = 4;
CALL PLITDLI (DLI_PARM_COUNT,
              GU_FUNC,
              CLAIM_PCB,
              ADDR(IOA_CLAIMANT_SEGMENT),
              CLAIMANT_SSA);
IF CLAIM_STATUS_CODE = ' '
THEN DO;
  TPO_ERRMSG1 = 'CLAIM NO ALREADY EXISTS';
  CONSIS_INPUT_ERRORS = 'E';
  TPO_CLMNO_ATTR = ERROR_ATTR;
  GO TO J_100_SELECT_RETURN;
END;
ELSE . . .
```
- For PL/I with PGMSTRUCT 2 or 3:


```
CLAIMANT_SSA_VALUE= XFER_CLAIM_NO;
DLI_PARM_COUNT = 4;
CALL PLITDLI (DLI_PARM_COUNT,
              GU_FUNC,
              CLAIM_PCB,
              ADDR(IOA_CLAIMANT_SEGMENT),
              CLAIMANT_SSA);
IF CLAIM_STATUS_CODE = ' '
THEN DO;
  TPO_ERRMSG1 = 'CLAIM NO ALREADY EXISTS';
  CONTROL_INDICATOR = DO_WRITE_LIT;
  TPO_CLMNO_ATTR = ERROR_ATTR;
  GO TO J_100_SELECT_RETURN;
END;
ELSE . . .
```

Attribute Considerations

This subsection presents considerations on the various attributes that you can specify for the fields on your screen. Become familiar with these attributes so that you can control such field characteristics as color and intensity. Topics included below are positioning of the cursor, standard attributes, extended attributes, other extended attributes, and error handling.

Cursor Positioning

CA Telon is concerned with cursor positioning at two times during screen processing:

- During output processing, before the screen is displayed. After CA Telon retrieves the data to be displayed for a screen and processes any output initialization logic (OINIT1 and OINIT2), it determines where the cursor is positioned when the screen is displayed.
- During input processing, after the generated field-edit logic has been performed. If CA Telon detects any edit errors, it positions the cursor to the first field in error when the screen is redisplayed for correction.

Topics discussed below include the key fields used to set the cursor, automatic processing to position the cursor, and considerations related to the specification of user-supplied cursor-positioning code.

Fields to set the cursor

There are three key fields used to position the cursor. By moving one of these fields to the attribute byte of a screen field, the CA Telon code essentially positions the cursor at the first byte of that field and displays the field in an intensity that corresponds to the key field used. (If more than one cursor attribute key field is used on the screen, the cursor is positioned to the first field with such an attribute setting.) The key fields are defined below:

- CURSOR-ATTR positions the cursor to the first character of the field and displays the field at normal intensity
- ERROR-ATTR positions the cursor to the first character of the field and displays the field at high intensity
- CURSOR-BLANK-ATTR positions the cursor to the first character of the field and suppresses the content of the field (for example, makes it nondisplay)

Automatic cursor positioning

There are two ways in which the CA Telon-generated code handles cursor positioning. (You need not use the automatic cursor positioning; user-defined positioning logic can be used instead, as described in this subsection.)

1. Initially, CA Telon sets the cursor to the field named after the CURSOR parameter (if you have defined the CURSOR parameter). To do this, it sets the named field's attribute to CURSOR-ATTR. If the field is in loop processing, CA Telon moves the cursor to the first occurrence of the field.

Note that you cannot position the cursor on a nondisplay field in this way (for example, to a field defined on the TDF as ATTRINT=blank); if this is necessary, you, the programmer, must position the cursor on field (using the CURSOR-BLANK-ATTR reserved field name).

2. When errors are detected during edit processing, CA Telon sets the cursor to the erroneous field. It does this by setting the attribute byte of the field in error to ERROR-ATTR. (If more than one field has errors, it moves the reserved name to each erroneous field; when the screen is displayed, the cursor appears in the first error field.)

Note that you cannot use generated edits for a nondisplay field, because CA Telon uses ERROR-ATTR (and not CURSOR-BLANK-ATTR) to set the cursor in the event of error. You must include all such edit code explicitly.

User-defined cursor positioning

You can control the positioning of the cursor, both initially and after edit processing, by following the steps below:

1. To define an initial cursor position, use the CURSCUS field on the Create/Update Screen Definition screen (rather than the same statement's CURSOR keyword, described above). After the CURSCUS field, include the name of the COPY member that contains the cursor-positioning logic to be used. CA Telon copies CURSCUS custom procedural code into section N-100-CURSOR-POSITIONING.

The code is executed just before the screen is output. Where there are select fields on the screen, the code is executed if either no option was selected or multiple options were selected.

Note: If any special I/O is necessary to determine cursor positioning, put it in section A-100-OUTPUT-INIT, using either the OINIT1 or OINIT2 exit described in Chapter 7, "Program Hierarchical Structure." You can save any intermediate results of the I/O in the transfer work area (named by the XFERWKA parameter described earlier in this chapter). Although you can do such I/O in the cursor-positioning custom code, it is less efficient than doing it in the A-100-OUTPUT-INIT processing; the custom code can be executed multiple times, whereas A-100-OUTPUT-INIT logic is executed only once.

2. To define cursor positioning following an edit, include your own code by using the FLDEDIT parameter.

Standard Attributes

There are two standard attributes that you can specify for the fields on your screens: ATTRPRO (protection attribute) and ATTRINT (intensity attribute).

- ATTRPRO—Use this attribute to specify if a field is to be protected. Normally it is used to protect an OUTIN field. No entry is allowed on a protected field.
- ATTRINT—Use this attribute to specify the intensity of the field to be displayed. Values are:
 - NORMAL : Normal intensity
 - HIGH : High intensity
 - BLANK : Not displayed

Overriding Attributes

Use the Out Attribute (OUTATTR) to specify whether the attributes for the field are to be included in the output buffer. Its purpose is to allow the attributes of a particular field to be overridden in custom code when the application is not intended to have attributes for all output fields included in the output buffer.

Note: OUTATTR is valid only for output fields and literal fields with field names. OUTATTR overrides the installation default for the specified field.

Extended Attributes

Extended attributes can be used only on an extended attributes terminal. You must set EATTR equal to **Y** on the Update Screen Parm's screen to use the following extended attributes.

- EACOLOR—Use this parameter to specify that an extended attributes terminal is being used and the color in which the field appears on the screen.
- EAHIGH—Use this parameter to specify that the field appears highlighted on the screen.
- EVALID—Use this parameter to specify that an extended attributes terminal is being used. This is only used when CA Telon is requested to generate IMS MFS source code.

Other Extended Attributes

In your generated code, you may notice an ERROR-F-ATTR and an ERROR-P-ATTR. These are attributes used by BMS that can be manipulated using custom code. The F attribute is the Field attribute; the P attribute is the Protect attribute. For information on these attributes, refer to your CICS Application Programmer's Reference Guide.

Error Handling

When errors are detected during edit processing, CA Telon sets the cursor to the erroneous field. It does this by setting the attribute byte of the field in error to ERROR-ATTR. If more than one field has errors, it moves the reserved name to each erroneous field.

When the screen is displayed, the cursor is positioned at the first error field. Note that you cannot use generated edits for a nondisplay field since CA Telon uses ERROR-ATTR (and not CURSOR-BLANK-ATTR) to set the cursor when errors occur. You must include all such edit code explicitly.

As illustrated by the above code, when an error is indicated to CA Telon in PFKEY or CONSYS processing, CA Telon assumes that you have used ERROR-ATTR to highlight the error. If the error is such that there is no field to highlight (for example, an invalid PF key was pressed), then the cursor must be positioned in custom code by performing N-100-CURSOR-POSITION.

Chapter 9: Processing Flow

This chapter covers several topics related to processing flow. Concepts covered include controlling processing flow, list processing, use of PF keys and SELECT fields, HELP/HOLD processing, IMS/DC report processing and line optimization considerations.

Subtopics under controlling processing flow include screen flow control options and internal program flow for main processing, output processing, and input processing.

Under list processing, subtopics include data positioning, restrictions, output processing, input processing, and data writing. Under the output processing subtopic, this subsection includes a discussion of browsing, paging, automatic line numbering, user-supplied processing, and using arrays.

General considerations on using PF keys, as well as information on using PF keys to control screen flow and for HELP/HOLD processing are then presented.

Subtopics under SELECT fields include control processing and flow with list screens.

HELP/HOLD processing includes PF key definition, screen statement requirements, database definition, variable storage, and text definition.

IMS/DC report processing includes report definition, report program structure, controlling the report destination, and calling program interface.

Line optimization considerations include an explanation of the parameters relevant to line optimization, such as LINEOPT and OUTIFIL.

Controlling Processing Flow

A particularly important concept is the transfer of control from one screen to the next; effectively, the transfer of control from one program to another.

There are several instances when you may want to transfer control to a different program. In some cases, the transfer is conditional; in other cases, it is not. Furthermore, if the transfer is conditional, where and how you actually pass control can depend on any of several factors: which field is selected; what PF key, PA key, or control key is pressed; or the outcome of processing logic.

CA Telon automates all three with the NEXTPGM, CONSYS, SCONSYS, or PFKEY parameters.

NEXTPGM Parameter

Using the NEXTPGM field, on the Create/Update Screen Definition screen you can specify explicitly the screen to which control passes after the processing for this screen is completed. Use this field only if the control passes unconditionally; if special (for example, conditional processing) code also specifies a next screen, the NEXTPGM specification takes precedence over subsequent special code.

The only instances in which control does not pass to the original screen specified by the NEXTPGM parameter are when:

- Errors occur during processing. In this case, the screen is redisplayed with an error message for correction.
- Explicitly coded instructions as described below are in a COPY member referenced for a PF key. PF-key logic is executed before the NEXTPGM logic and can circumvent the latter's processing.

Using the NEXTPGM field (as defined on the Update Select Field screen) for a field defined as SELECT, you can explicitly specify the screen to which control passes after the select preprocessing for the field. Use this parameter only if the control passes unconditionally when this field is selected; if special (for example, conditional processing) code for the field also specifies a next screen or if SELECT fields are coded, the NEXTPGM specification is ignored.

Note that if errors occur during select-field or PF-key processing, control does not pass to the screen identified by the NEXTPGM parameter; rather, it returns to the current screen.

CON SIS Parameter

Using the CON SIS field defined on the Panel Definition menu or the Create/Update Screen Definition screen, you can specify the name of the COPY code that processes screen-level consistency logic. Within the CON SIS code, you can include a statement such as that shown below to define the screen to which control passes after the current screen is processed.

- For COBOL:

```
MOVE 'screen-id' TO NEXT-PROGRAM-NAME-ID.
```
- For PL/I:

```
NEXT_PROGRAM_NAME_ID = 'screen-id';
```

where screen-id identifies the ID of the screen to receive control, either as the ID itself, enclosed in single quotes, or as the name of a variable field that contains the screen ID.

SCONSIS Parameter

Using the SCONSIS field (as defined on the Update Select Field screen) for a field defined as SELECT, you can specify the name of the COPY code that processes logic associated with the field, when it is selected. You can include a statement such as that shown above, within the CONSIG code, to define the screen to which control passes after the SCONSIS selection processing.

PFKEY Parameter

Using the PFKEY field defined on the Create/Update Screen Definition screen, you can specify the name of the COPY code to process each different PF key that can be selected. To control which screen gains control using the PF-key processing logic, you can, within this logic:

- Request transfer of control to the next program. If you are using mainline option 1, request transfer by setting PFKEY-RETURN-INDICATOR to **R**. If you are using mainline option 3, request transfer by setting CONTROL-INDICATOR to DO-TRANSFER-LIT
- Move the ID of the next screen to be processed into the reserved field, NEXT-PROGRAM-NAME-ID (Sample code to do this is shown earlier, for COBOL and PL/I.)

List Processing

List or loop-processing screens include a group of fields duplicated multiple times on a screen. For the same field appearing in multiple groups, the column position remains the same, but the row position varies. The following example illustrates a list screen where the last 13 lines (shown as essentially blank lines above the select option) are processed using CA Telon's iterative loop handling.

```
PAGE >> >>>>>
      ILLUSTRATE THE GENERATED SEGLOOP CODE FOR DATA BASES

      XFER SAVE FIELD ++++++

      LINE      SEGMENT KEY      THE ONLY FIELD IN THE SEGMENT
      >>      ++++++++      ++++++++
```

List screens are implemented using the SEGLOOP processing defined on the Panel Definition menu. You can have only one SEGLOOP per screen.

SEGLOOP Processing

SEGLLOOP processing can be implemented on both output and input; that is, multiple groups of the same fields can be displayed, entered, or displayed and then entered on a screen. On each SEGLLOOP, you indicate the usage for the loop by identifying it as input, output, or outin. Note, however, that the characteristics differ significantly between output and input loop processing.

This subsection discusses general considerations related to SEGLOOP processing. It includes the following:

- Positioning in a SEGLOOP
- Restrictions on SEGLOOP processing
- Output and input loop processing
- Data writing

Vertical Data Positioning

Data that is repeated vertically (in rows) in loop processing on the screen is positioned as follows.

The first iteration of the group is positioned at the row and column specified by the LINE and COLUMN field on the Create/Update Table SEGLOOP screen.

Subsequent iterations are positioned below the first, aligned in the same starting columns. The starting row for each iteration (after the first) is determined by the INCRE field on the Create/Update Table SEGLOOP screen. The INCRE field specifies one number for each iteration of the loop on the screen, after the first, thereby controlling the number of iterations processed by the screen.

INCRE=(1,1,1,1,1,1,1,1,1,1,1)

Each number corresponds positionally to an iteration of the loop. When the screen is displayed, the starting row of each iteration is offset from the starting row of the previous iteration by the number of rows in its corresponding INCRE specification.

For example, with INCRE=**1,1,1**, four rows occur. The first line is painted by you. The second occurs one line down from the first line. The third occurs one line down from the second line. The fourth occurs one line down from the third.

With INCRE=**1,2,1,2,1**, six rows occur. The first line is painted by you. The second occurs one line down from the first line. The third occurs two lines down from the second line. The fourth occurs one line down from the third line. The fifth occurs two lines down from the fourth line. The sixth occurs one line down from the fifth line.

Horizontal Data Positioning

Data that is repeated horizontally (in columns) in loop processing on the screen is positioned as follows.

The first iteration of the group is positioned at the row and column specified by the LINE and COLUMN parameters, the same as for INCRE earlier.

Subsequent iterations are positioned to the right of the first according to the various CINCRE values (the same as for INCRE except repeated horizontally instead of vertically). For example, with INCRE=**1,1,1** and CINCRE=**25,25**, you get three columns of four rows each. CINCRE=**25,25** results in three columns. The first is painted by you. The second occurs 25 characters to the right of the beginning of the first column. The third begins 25 characters to the right of the beginning of the second column.

CICS programs that use BMS can use CINCRE only if the SEGLOOP is a single line SEGLOOP. CICS programs not using BMS can use multi-line SEGLOOPS with CINCRE provided all fields are OUTPUT fields or all OUTIN or INPUT fields appear on a single line in the SEGLOOP.

For Non-BMS CICS Programs

Note: In the Prototyping Facility, if OUTIN or INPUT fields appear on more than one line, CINCRE is ignored and only the left-most column is displayed. All other columns are filled with nulls and not displayed.

Restrictions

There are two restrictions that fall within a loop. They are:

- The field-names assigned to those fields cannot exceed six characters in length. (Normally, field-names can be up to eight characters.)
- LITERAL fields and SELECT fields cannot be included in a loop. All LITERAL FIELDS precede variable FIELDS and, therefore, cannot fall in the SEGLOOP sequence. If you want to display a literal in a loop, you must define the field as an OUTPUT field and either map it to the screen from a field that contains the literal value or move it to the output buffer directly, through user-defined code.

Output Processing

List screens are commonly used on output to display multiple iterations of the same type of data either from multiple segments of the same type (for DL/I processing), from multiple records in the same file (for VSAM processing), or from an array (for DL/I or VSAM-file processing). Frequently, along with the data listed on output, the screen includes a select field used by the application user to indicate which of the listed items are to be processed further. This concept is discussed further later in this chapter.

The rest of this subsection covers loop processing under the following topics:

- Browsing – An introduction to the I/O involved and the ways in which you can request browsing
- Paging – The options available to you when the list takes more than one screen to display, and related processing
- Automatic line numbering – A description of CA Telon's facility by which you can number listed items and, optionally, tie them to a select field used to choose one of the items for subsequent processing
- User-supplied processing – A discussion of when and how you can insert user-specific code related to loop processing
- Using arrays – Information related to the option of writing the list from an array, rather than from the data files directly

Browsing

The I/O involved with loop processing is known as *browsing*. You tell CA Telon that you intend to browse a file when you specify a file access of REQUEST on the Create/Update Data Group screen. To browse through a file, you start at one point in a file and move through it, looking for records that are candidates to be included in the list on the screen. In some cases of DL/I access, the candidates are all occurrences of the same segment type under a particular parent record; in other cases, the candidates are differentiated logically.

You can let CA Telon generate your browse I/O automatically, or you can handle it in custom code. Use a BROWSE REQUEST to indicate whether you want CA Telon to generate the I/O. If you do not specify a BROWSE REQUEST, you must include any required I/O in the COPY member referenced by the OCUST1 and OCUST2 parameters.

There are two mutually exclusive ways for you to tell CA Telon where you want to start the browse. If you do not use either method, the browse begins at the first occurrence of the segment under which parentage has been established (for DL/I files) or at the first record in the file (for VSAM files). In either case, the browse continues until there are no more segments/records to search or until the requested number of segments/records, defined using the INCRE parameter, have been processed.

To browse starting at the first record having a particular key value, supply the STBRKEY field, on the Create/Update File SEGLOOP screen, to define the field that contains the start-browse key at run time. For VSAM, you can supply either a full or partial key value in STBRKEY; if you supply a partial value, the browse starts with the first segment/record whose key value(s) matches the characters specified, for the number of characters specified.

In earlier pre-2.x releases of CA Telon, use of SCHFLDx fields were permitted, as follows:

For DL/I file access only, to browse based on a specific field value in the segment (generally, to ensure that all records displayed have the same value for a particular field; SEX=**M**, for example), use the SCHFLDx field on the Create/Update File SEGLOOP screen (described later in this chapter), and set the OPCODE field on the Update Database Segment screen as follows. Since this type of search is unkeyed, do not use it on the root level of a database, unless the database is very small.

- SCHFLDC is the data item that contains the value that the record's SCHFLDI item must be equal to in order to be displayed on the screen.
- SCHFLDI is the name of the field, as defined in the DL/I Database Definition (DBD), against which the SEGLOOP search is directed. The segments listed on the screen are those for which the SCHFLDI value matches the value in SCHFLDC.

- SCHFLDL is the length of the fields named by the SCHFLDC and SCHFLDI. These fields must be equal in length.
- OPCODE, on the Update Database Segment screen, is used to indicate the operation desired. For browse processing, it is automatically set to greater-than or equal-to (>=). When you use the SCHFLDx fields, set the OPCODE as is appropriate. Generally, to display records that have the same value for a particular field, you set the OPCODE to equal (= for DL/I), although your logic dictates that you leave it as greater-than or equal-to.

Note: The SCHFLDx fields are only displayed on the Update SEGLOOP (P170 & P175) screens for programs created with a release prior to 2.0.

In the following examples, all DL/I segments having a value, in the field named SEX, equal to the value in the field named XFER-SEARCH-SEX will be displayed. For this search to work correctly, the OPCODE must be set as follows:
OPCODE='='.

```
XXXXXX.UPDATE DATABASE SEGMENT *****
UPDATE DBD: _____ SEGM: _____
COMMAND ==> _____
OPTIONS ==> PCB PARMS

GENERAL LABEL _____ USAGE BROWSE KEYPIC _____
*      COPY _____ COPYLV1 COPYLBL _____
*      KEYLEN 4

**      DSCREF SEGMENT CMND IMSKEY OP KEY

--      _____ SSIRMKEY = _____
```

Note: The use of SCHFLDx processing is limited to carry-overs from pre-2.x releases of CA Telon. No new or altered specifications of these fields are possible.

```

XXXXXXXX.PD UPDATE FILE SEGLOOP *****
COMMAND ==>
      OUTPUT SEGLOOP  LIMITS NAME      LINE      COLUMN
                        FIRST SEQ__ 007      003
                        LAST ADDR2_ 010      007

INCRE 1 1 2 1 1
REPEAT 1
CINCRE
LINECNT LINENO
PAGE Y PAGESAV 120 PKYUNIQ _ PKYLTH
PAGEKEY ROOM-KEY
OCUST1 OUT1 OCUST2 OUT2 OCUST3
OSEGIDX
SAVEKEY SEGMENT-KEY TO SAVE-SEGMENT-KEY
      TO
      TO
      TO
      TO
      TO
      TO
STBRKEY XFER-ROOM-KEY ISEGIDX
ICUST1 ICUST2 ICTLNM

```

Paging

In some cases, all data available for output cannot fit on a single screen. To allow you the maximum control over such situations, each output (OUTPUT or OUTIN) loop screen is either allowed or not allowed to display the data that does not fit on the first screen.

To use the output loop screen, use the PAGE field on the Create/Update File SEGLOOP screen. If PAGE=**N**, only one screen of data is displayed, even if more data exists in storage. If PAGE=**Y**, however, all available data is displayed, using as many pages as necessary. Each page is numbered and, when more data exists, the message ***MORE is displayed.

You can page forward and backward using PF keys defined for this purpose. However, unless you specify a number (up to 999) of back-pages to save using the PAGESAV field on the Create/Update File SEGLOOP screen, the generated code saves only one page. If you scroll back beyond the previous page, you start again at page 1.

If PAGE=**Y**, you must define the PAGENO and MORE control fields somewhere on the screen, but not in the SEGLOOP iterations. These fields are used, respectively, to display the page number and ***MORE message.

The screen designed for loop processing, shown earlier in this chapter, shows the PAGENO and MORE fields in the upper-right corner.

Also, if PAGE=Y, there are special requirements to include related fields in the transfer work area. These fields, LAST-LINENO and PAGE-AREA-START, are used, respectively, to store the number of the last line written and paging information.

Note: No paging logic is generated unless PAGE=Y and an automatic BROWSE request is used.

Paging for BOOLEAN DSCs

The SEGLOOP Browse Facility in CA Telon allows you to specify BOOLEAN DSCs. You can reference these BOOLEAN DSCs at both the UIO level and at higher levels, if desired. When there is a multi-level BROWSE and paging, you *must* specify the length of the concatenated key for the UIO in the SEGLOOP, using the PKYLTH parameter. This is because a concatenated key call is generated for pages other than page one.

There are also recognizable differences in the source code generated for a SEGLOOP when you process using a BOOLEAN DSC. The major differences are as follows:

- The structure of SEGLOOP-2-SSA
- The generated *paging* logic in B-100-OUTPUT-SEGLOOP-INIT
- The use of IGNORE conditions in the *paging* logic

The SEGLOOP-2-SSA for a BOOLEAN DSC is always generated as a concatenated key SSA.

The *paging* logic with a BOOLEAN search argument is noticeably different from a non-BOOLEAN search argument. Because CA Telon generates a concatenated key call to obtain the first item on the next page, the program looks for one specific record on the database. If the record is no longer on the database, the user is returned to page 1 of the SEGLOOP. For non-BOOLEAN DSCs, CA Telon generates an SSA that looks for records that are greater than or equal to the last record on the page, so that it can get the first item on the next page. Because the SSA is not looking for a specific record, it does not follow the same logic as the concatenated key SSA.

Note: IGNORE conditions are not in effect for the BOOLEAN *paging* mechanism, because there is no override to the *return to Page 1* routine. The length of the CURRENT-SEGMENT-KEY (or CURRENT_SEGMENT_KEY for PL/I) must be the length of the full concatenated key to provide for proper positioning in the database. It is recommended that the key feedback area for the database be used as the PAGEKEY.

PATH can be specified if data is to be retrieved from higher level segments.

Note: When non-keyed segments are at levels above the UIO level, they are not supported.

Paging on a DL/I segment using KEYPIC

When you execute an autoexec browse against a DL/I segment that has KEYPIC on the SEGMENT key, you should be aware of how the paging variables are defined in the CA Telon COBOL or PL/I program.

When you have unique keys (PKYUNIQ = Y) and are not using a BOOLEAN DSC, the following variables will have the same definition as the SEGMENT key (which is defined using KEYPIC):

- SEGLOOP-1-SSA-VALUE
- SEGLOOP-2-SSA-VALUE
- CURRENT-SEGMENT-KEY
- PAGE-KEY-SAVE
- START-BROWSE-KEY

When you have non-unique keys (PKYUNIQ = N) and are not using a BOOLEAN DSC, the following variables will have the same definition as the SEGMENT key (which is defined using KEYPIC):

- SEGLOOP-1-SSA-VALUE
- SEGLOOP-2-SSA-VALUE
- CK-SEGMENT-KEY
- PK-SEGMENT-KEY
- START-BROWSE-KEY

The following variables are defined as alphanumeric or character if you have non-unique keys:

- PAGE-KEY-SAVE
- CURRENT-SEGMENT-KEY

CA Telon SEGLOOP parameters, such as PAGEKEY, STBRKEY, and SAVEKEY, should be chosen with the above facts in mind.

Automatic Line Numbering

For your convenience, CA Telon offers a facility that numbers each group of fields (for example, each loop iteration) automatically, starting with 1 on each page. To request this facility, include the LINECNT field on the Create/Update File SEGLOOP screen, setting LINECNT to the name of the field in which you want the automatically generated number to appear. The target field for the LINECNT must be defined as a one- or two-digit numeric field (PIC '9', PIC '99', or PIC 'Z9') and must have a FLDTYPE of **NONE**, as illustrated below.

```

XXXXXX.PD UPDATE OUTPUT FIELD *****
COMMAND ==>
FIELD NAME HHHH __ USAGE OUTPUT __ LINE 02 COL 002 LTH 001

GENERAL IN: REQ (Y/N/C) HELPMMSG _____
* OUT: PIC 9 _____ OUTATTR (Y/N)

MAPPING: DBNAME _____
*          OF _____
*          .....
*          OF _____
*          .....
*          OF _____
*          INIT _____
*          MAPOUT _____

EDIT:  FLDTYPE OUT NONE __ IN __ PARM LIST EXTENSION
*      SPEC _____ (FORMAT/CONVERT)
*      _____
*      _____
*      _____
*      _____

ATTR:  ATTRPRO __ ATTRINT __ EACOLOR __ EAHIGH __ EVALIDD __
       FMTEXTIT _____ FMTONTL=MFS __ (Y/N)

```

Note: If you number the fields by defining the line numbers as literals, then all numbers are always displayed, regardless of how many iterations of the loop are displayed.

An additional capability, generally used together with the LINECNT facility, is the automatic tie-in to a SELECT field in which the application user enters the line number of the item for which he/she wishes additional processing. Using the entered line number, the system passes the key (or other unique) value for the line item to another screen—usually to display more data for the line item on that screen.

CA Telon's facility of saving, in a table, the key data for each iteration displayed on a list screen makes this easy. This ensures that key data is available during select processing.

To request that all key fields for the displayed segments/records be saved, use the SAVEKEY field on the Create/Update File SEGLOOP screen. For information on how to specify the arguments for the SAVEKEY parameter, refer to the *Design Facility Reference*.

Briefly, you define, for each key component to be saved (per iteration), the key value as defined and the name of the field, in the save-key table, where the value is to be stored. To use this option, the save-key table must have been established in the transfer work area (named by the XFERWKA parameter on the Create/Update Screen Definition screen). The following code illustrates such a table:

- For COBOL:

```
05 XFER-SEGLOOP-TABLE OCCURS 13  
10 XFER-TABLE-MSG-KEY PIC X(9).
```

- For PL/I:

```
05 XFER_SEGLOOP_TABLE(13),  
10 XFER_TABLE_MSG_KEY CHAR(9);
```

Once the save-key table is established, you can define a SELECT field for use with the automatic tie-in to the screen line numbers. To do this, use the SELKEY field on the Update Select Field screen for the select field to define:

- The data item in which the key value(s) is stored, or the displayed item that corresponds to the line selected
- The data item in which you want to store that key value(s) for processing by the next screen

For more information on the SELKEY parameter, refer to the *Design Facility Reference*.

At run time, when you have specified the SELKEY information and the application user enters a line number in the corresponding SELECT field on the screen, the following processing is done:

- The line number entered is edited to ensure that it specifies a valid line number and one for which data is displayed.

If an error is found, an error message is returned to the screen and the processing logic awaits subsequent correction by the user.
- The key value in the save-key table for the line number entered is moved to the field defined by the second argument of the SELKEY parameter. Note that this field, as well as the SAVEKEY table, must be defined in the transfer work area.
- If there is a NEXTPGM defined, control passes to the processing for the next screen as defined by NEXTPGM; otherwise, it passes to the next screen as defined in SCONSIS logic. (Either way, a next screen must have been supplied.)

The following example illustrates the SELKEY parameter used to enter a line number that the application user wants to process further.

```
XXXXXX.PD UPDATE SELECT FIELD *****
COMMAND ==>
FIELD NAME LINE_____ USAGE SELECT LINE 23 COL 21 LTH 002

GENERAL: NEXTPGM MZXX SCON SIS _____ HELPM SG _____
* INEDIT (Y/N) INDBIO (Y/N)
* SELKEY FROM XFER-TABLE-MSG-KEY
* TO XFER-MSG-KEY

MAPPING: DBNAME1 _____
*          OF _____
*          DBNAME2 _____
*          OF _____
*          INIT _____

EDIT:  FLDTYPE _____ PARM LIST EXTENSION
*      SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*
*
*
*
ATTR:  ATTRPRO ATTRINT _____ EACOLOR EAHIGH EAVALID
       FMTEXTIT __ __ FMTCNTL=MFS (Y/N)
```

Here, when the application user enters a valid number in the select field (named LINE), the system moves the key from the save-key table (XFER-TABLE-MSG-KEY) to XFER-MSG-KEY, and passes control to the next program, MZXX. MZXX presumably displays information for the record whose key value(s) is stored in XFER-MSG-KEY to begin its processing.

User-Supplied Processing

In some cases, as mentioned previously, you need to use special (for example, non-generated) processing logic. In such instances, you can include supplemental or replacement I/O logic using the OCUST1, OCUST2, and OCUST3 fields on the Create/Update Table SEGLOOP and Create/Update File SEGLOOP screens. For a description of these parameters, refer to the *Design Facility Reference*.

Reasons why you might want to include user-defined I/O include:

- Additional I/O, to include processing to read additional segments/records; frequently, for DL/I processing, a child of the segment already read. (Standard SEGLOOP processing automatically handles the I/O for a single type of segment or record.)
- I/O analysis, to test the data retrieved through the auto exec for particular conditions before moving it to the screen (for example, to compare secondary fields to see if they meet necessary criteria).

- Replacement I/O, to substitute entirely for the auto exec, in the event that the user exec is inadequate.
- SEGLOOP exit analysis, to handle loop termination when the generated processing is not sufficient.
- Output field editing, to handle those fields that cannot be moved to the screen using standard output edit logic.

If you request user-specific code using any of the OCUSTn parameters, it is positioned in the generated source code, as described in Chapter 13, "Application System Generator". A summary of this information follows:

- OCUST1 code is added just after the auto exec call for the first loop-screen item and before the screen mapping for that item. It is bypassed if the auto exec cannot retrieve a segment/record successfully. OCUST1, for the first loop-screen item, is the equivalent of OCUST2 plus OCUST3 for all subsequent iterations.
- OCUST2 code is added just after the subsequent I/O call, which retrieves the next item to be listed. It is usually used to suppress unwanted segments returned by the auto exec, or to do custom I/O when the auto exec has been suppressed. OCUST2 logic is bypassed if the auto exec cannot retrieve a segment/record successfully.
- OCUST3 code is added at the end of processing for each iteration of the loop, before mapping the fields for display. OCUST3 follows a check to see if there is room on the screen for the item last retrieved (and, if applicable, processed through OCUST2). If there is room for the item, OCUST3 is executed. Usually, OCUST3 logic is used for special output edits, just before field mapping at the beginning of the next loop iteration; therefore, it usually contains some of the same code as OCUST1.

Using Arrays

Sometimes the data in a list is not moved directly from a segment or VSAM record, but rather from a user-defined indexed table or array. In such cases, a read prior to the move to the screen is not required, and the considerations discussed above do not necessarily apply.

When mapping from an array to the screen, you must define and fill the array with data, either in a prior program or in the output initialization code of the current program. Other considerations follow:

- Paging cannot be handled automatically. You must set the PAGE parameter to **N**. To include paging logic, you must do so in OCUST1 or OCUST2 custom code. For consistency with CA Telon's methodology, use the TPO-MORE field. To determine what PF key has been pressed by the application user at any point in time, use the PFKEY-INDICATOR reserved field.
- You must not include a BROWSE segment.

- While most keywords discussed previously in this chapter apply, the PAGESAV, SEGKEY, OPCODE, SAVEKEY, STBRKEY, and SCHFLDx parameters do not apply and are ignored.
- On the Update Select Field screen, include for each array field, the name or subscript field (SEGLOOP-COUNT) in the DBNAME for the field, as illustrated for a COBOL application in Update Select Field Screen - Example 1 and Update Select Field Screen - Example 2. The first example assumes that OSEGIDX=**TBL-INDEX** assumes the use of a subscript.

```

XXX.PD UPDATE SELECT FIELD *****
COMMAND ==>
FIELD NAME LINE_____ USAGE SELECT LINE 23 COL 21 LTH 002

GENERAL: NEXTPGM MZXX SCON SIS _____ HELPM SG _____
* INEDIT (Y/N) INDBIO (Y/N)
* SELKEY FROM _____
* TO _____

MAPPING: DBNAME1 MSG-KEY
*          OF TBL-INDEX
*          DBNAME2 _____
*          OF _____
*          INIT _____

EDIT:     FLDTYPE _____ PARM LIST EXTENSION
*         SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*
*
*
*
ATTR:     ATTRPRO ATTRINT _____ EACOLOR EAHIGH EVALID
          FMTEXTIT __ __ FMTCNTL=MFS (Y/N)

```

```

XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME LINE_____ USAGE SELECT LINE 23 COL 21 LTH 002

GENERAL: NEXTPGM_____ SCONSIS_____ HELPMMSG_____
*      INEDIT (Y/N) INDBIO (Y/N)
*      SELKEY FROM_____
*      TO_____

MAPPING: DBNAME1 MSG-KEY
*      OF SEGLOOP-COUNT
*      DBNAME2_____
*      OF_____
*      INIT_____

EDIT:   FLDTYPE_____ PARM LIST EXTENSION
*      SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*
*
*
*
ATTR:   ATTRPRO ATTRINT_____ EACOLOR EAHIGH EAVALID
        FMTEXTIT __ __ FMTCNTL=MFS (Y/N)

```

- For every other reference to an item in the array, you must also include a subscript or, for COBOL only, an index. When using a subscript, include the reserved-field name, SEGLOOP-COUNT, as a subscript when the array fields are referenced.
- The generated logic assumes that each entry in the array must be mapped. If this is not the case, include the code to exit the loop in the OCUST2 user-specified logic.

Input Processing

You can use list screens for input as well as output. When a list screen is used for input, it allows you to add or change data for multiple iterations of the same segment or record type. There are two approaches to this:

- Edit and move the data for each loop iteration to an I/O area and, from there, write the data at the end of each pass through the loop. This technique is more straight-forward and mirrors that used during SEGLOOP output processing. However, it has the drawback of not allowing you to edit all the input fields together before the I/O is performed.
- Edit and move the data for each loop iteration to an array. Save the data in the array until all input has been edited individually. Then, using the consistency edits, perform any cross-field checks and write the array to the appropriate segments/records. (If the array is part of the transfer work area, you can wait until the next program to write the array.)

Note: Of the two techniques, the second is generally preferable.

To accommodate input processing, there are three SEGLOOP parameters and two reserved fields for use with input-screen processing. These new parameters/fields are discussed next, followed by considerations specific to the two techniques of processing input loop screens.

Parameters

The three SEGLOOP parameters for use with input-screen processing follow :

- ICTLNM identifies a field in the SEGLOOP sequence that (if the field contains spaces for a loop occurrence) causes SEGLOOP field editing to be skipped. You can use it in combination with ICUST2 code, described next, to signal the end of loop processing.
- ICUST2 identifies a section of special processing code that is executed for each pass through the loop. If you want to use the ICTLNM field, described above, to terminate loop processing, include the appropriate check in ICUST2 logic.
- ISEGIDX, applicable when using an indexed array under COBOL, defines the index name for the array.

Reserved Fields

The two reserved fields used for input processing are INPUT-LINE-EDIT and INPUT-LINE-COUNT, described below :

- INPUT-LINE-EDIT provides a status of the screen line last processed. It is set as follows:
 - N (COBOL 88-level NO-LINE-EDIT) indicates that the field editing for this loop iteration was skipped (described above for the ICTLNM parameter).
 - E (COBOL 88-level LINE-ERRORS) indicates that there was a field error in processing the loop iteration.
 - Spaces (COBOL 88-level NO-LINE-ERRORS) indicates that the iteration was processed successfully (for example, no field edit errors were detected).
- INPUT-LINE-COUNT maintains a count of the lines processed in the loop. You can use it as a subscript for the input array if you use the arrayed approach to processing the screen.

Data Writing from an I/O Area

When you write the data to your files for each pass through the loop, you must include the logic to write the data in the ICUST2 logic. To signal the end of input data, use the ICTLNM parameter specification, described earlier. (Otherwise, processing continues for the number of iterations defined by the INCRE parameter.)

Data Writing from an Array

If you store the screen-input data, to be written later, in an array, you can include the logic to write the data in any of three places:

- In CONSYS logic (defined on the Create/Update Screen Definition screen), processed after the loop processing is completed
- In SCONSYS logic (defined on the Update Select Field screen), processed after the loop processing is completed
- In a subsequent screen, assuming the array is stored in the transfer work area

There are two design considerations that apply to the arrayed technique of input-screen processing:

- The array to which the data is mapped must be set up in working storage or the transfer work area. It is easiest if this area has the same format as the segment or record to which the fields are eventually moved, thereby allowing all data for a loop iteration to be stored using a single move.

- For each array field (for SELECT fields), include the index name or subscript field (SEGLOOP-COUNT) in the DBNAME for the field, as illustrated for a COBOL application in Update Select Field Screen - Example 1 Update Select Field Screen - Example 2. The first screen example assumes that, on the SEGLOOP, ISEGIDX=**TBL-INDEX**. The second example assumes use of a subscript.

Input Field Editing for Segloop Fields

CA Telon generated code performs field edits for segloop fields in a loop in the E-100-INPUT-SEGLOOP paragraph (for PL/I, in the E_100_INPUT_SEGLOOP proc). For information on how to generate XFEDITS and SEGEDITS that are performed in a loop for segloop fields, see the description of consistency edits in section 8 of this manual.

```

XXXXXX.PD UPDATE SELECT FIELD *****
COMMAND ==>
FIELD NAME LINE_____ USAGE SELECT LINE 23 COL 21 LTH 002

GENERAL: NEXTPGM MZXX SCONSIS_____ HELPMMSG _____
*      INEDIT (Y/N) INDBIO (Y/N)
*      SELKEY FROM _____
*      TO _____

MAPPING: DBNAME1 MSG-KEY
*      OF TBL-INDEX
*      DBNAME2 _____
*      OF _____
*      INIT _____

EDIT:   FLDTYPE _____ PARM LIST EXTENSION
*      SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*      _____
*      _____
*      _____

ATTR:   ATTRPRO ATTRINT_____ EACOLOR EAHIGH EVALID
        FMTEXTIT __ __ FMTCNTL=MFS (Y/N)

```



```

XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME LINE_____ USAGE SELECT LINE 23 COL 21 LTH 002

GENERAL: NEXTPGM MZXX SCON SIS_____ HELPM SG _____
*        INEDIT (Y/N) INDBIO (Y/N)
*        SELKEY FROM _____
*        TO _____

MAPPING: DBNAME1 MSG-KEY
*         OF SELOOP-COUNT
*         DBNAME2 _____
*         OF _____
*         INIT _____

EDIT:    FLDTYPE _____ PARM LIST EXTENSION
*        SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*
*
*
*
ATTR: ATTRPRO ATTRINT _____ EACOLOR EAHIGH EAAVALID
FMTEXTIT __ __ FMTCTL=MFS (Y/N)

```

- For every other reference to an item in the array, you must also include a subscript or, for COBOL only, an index. When using a subscript, include either of the reserved-field names, SELOOP-COUNT or INPUT-LINE-COUNT, as a subscript when the array fields are referenced.

The following example illustrates SELOOP coding for arrayed input screen loop processing:

```

XXXXXX.PD UPDATE FILE SELOOP ***** *****
COMMAND ==>
OUTPUT SELOOP
LIMITS NAME LINE COLUMN
FIRST SEQ__ 007 003
LAST ADDR2_ 010 007

INCRE 1 1 2 1 1 _____
REPEAT 1 _____
CINCRE _____
LINECNT LINENO
PAGE Y PAGESAV 120 PKYUNIQ _ PKYLTH ____
PAGEKEY
ROOM-KEY
OCUST1 OCUST__ OCUST2 _____ OCUST3 OCUST3__
OSEGIDX
SAVEKEY SEGMENT-KEY_____ TO SAVE-SEGMENT-KEY_____
_____ TO _____
_____ TO _____
_____ TO _____
_____ TO _____
_____ TO _____
STBRKEY XFER-ROOM-KEY_____ ISEGIDX _____
ICUST1 _____ ICUST2 _____ ICTLNM _____

```

PF Keys

As with other online systems, applications developed under CA Telon offer two ways in which the application user can control the flow of processing in the application:

- Through PF-key selection. On any given screen, the application user can be asked to press a particular PF key to request a processing option defined on that screen. Depending on the PF key pressed, program logic proceeds accordingly.
- Through selection fields. As an alternative to PF-key selection, the application user can be asked to enter a value in a SELECT field that determines the processing that follows. For example, to request either an update or add operation against an accounts receivable file, the application user can enter an invoice number next to either an Add SELECT-field or Update SELECT-field prompt. Not only is the application user thereby providing the target invoice to be processed, but also determining what type of processing follows (specifically, in this case, add or update processing).

This subsection discusses the first of these two ways to control the flow of application processing. Discussion of control through the use of selection fields follows afterwards. Specifically, this subsection is broken down as follows:

- Overview of PF-key determination—How you determine which PF key was pressed from the screen
- Requesting PF-key processing—How you fill out the non-procedural statements to request automatic PF-key processing
- PF-key processing conventions—Instructions used when coding user-defined logic to handle PF-key processing

To Control Screen Flow

As noted above, PF keys are generally used in online applications to direct the flow of processing. You can always check the reserved field, PFKEY-INDICATOR, to determine which control key was pressed on the last screen displayed; the indicator lets you know which of the PF keys, PA keys, Enter, or Clear keys was pressed. For COBOL, the PFKEY-INDICATOR field definition includes 88-level items that can be used for ease of reference (for example, 88 ENTER-KEY; 88 PFK1).

On occasion, however, you must trap PF-key selection when the screen is first transmitted—before the mainline processing logic is entered. As an example, suppose that one PF key is used to request that processing be aborted. If that key is pressed, the logic must know to exit immediately with no further processing.

To accommodate these situations, in the input processing area, CA Telon provides exits that can specify user-defined logic to be performed depending on which PF key was pressed. In P-100-PFKEYS processing, the PF-key exit logic is called immediately after the screen is received.

Requesting PF-Key Processing

To define PF-key processing, include the following in the TDF:

- On the Create/Update Screen Definition screen, indicate which PF keys are used by the screen (for example, PFKEYS=(1,2,9,OTH) for PF keys 1, 2, and 9, and all Other PF keys, respectively).
- For each PF key specified, make the appropriate COPY code accessible to the program, under a name that follows the conventions of your installation. As installed, the name must be hhPFKnnn, where hh is the screen header and nnn is a one- to three-character code that identifies the PF key(s) processed by the code.

For the example above, member names hhPFK1, hhPFK2, hhPFK9, and hhPFKOTH are included in the program code. Also, the logic for each of the first three members processes PF-key 1, 2, and 9, respectively, and the code for the last one processes all other PF-key selections (although this need not be the case).

At run time, if an application user presses a PF key not handled by the custom code brought in via the PFKEY keyword statement, it is treated as an Enter.

PF-key Processing Conventions

Generally, you can do any type of processing in the PF-key code. However, your PF-key routines must conform to the following three conventions:

1. Each routine must begin with a check to see if the specified PF key was pressed (for example, IF PFKEY-INDICATOR = 1... or IF PFK1...). This allows you to use the same code to check for multiple PF keys, if desired. You might, for example, have a single member, hhPFK1#3, that processes for PF keys 1, 2, and 3.

2. Before exiting from the mainline routine, the PF-key routine must indicate the action for the mainline routine to take, depending on whether you are using mainline option 1, 2, or 3:

If using mainline option 1, do one of the following three actions:

- Set the PFKEY-RETURN-INDICATOR to **R** to terminate processing and pass control to the program identified by the reserved field, NEXT-PROGRAM-NAME-ID.
- Set the PFKEY-RETURN-INDICATOR to **E** to terminate input processing and send the input screen back to the terminal user. Move a message to the reserved field, TPO-ERRMSG1, to identify the problem or describe the activity performed.
- Set the PFKEY-RETURN-INDICATOR to blank to continue normal input processing. While this is unusual, it can be used to vary the processing if the PF-key processing parallels that for the Enter key, possibly with some additional logic up front.

If using mainline option 2, you can either take those actions explained above under mainline option 1 or those explained below under mainline option 3.

If using mainline option 3, do one of the following three actions:

- Set CONTROL-INDICATOR to DO-TRANSFER-LIT to terminate processing and pass control to the program identified by the reserved field, NEXT-PROGRAM-NAME-ID.
- Set CONTROL-INDICATOR to DO-WRITE-LIT to terminate input processing and send the input screen back to the terminal user. Move a message to the reserved field, TPO-ERRMSG1, to identify the problem or describe the activity performed.
- Do not modify CONTROL-INDICATOR to continue normal input processing. While this is unusual, it can be used to vary the processing if the PF-key processing parallels that for the Enter key (possibly with some additional logic up front).
- After PF-key processing, branch to P-100-PFKEYS-RETURN, the last statement in the generated P-100-PFKEYS processing logic.

The following code illustrates PF-key processing for PF1 for COBOL and PL/I with each mainline option. Although the COPY-member name is hhPFK1, it need not process PF1. (but using hhPFK1 to process other PF keys probably violates your installation naming conventions).

- For COBOL with mainline option 1:

```
.
. (remarks that describe processing)
.
IF PFK1
.
. (processing logic goes here)
.
MOVE 'R' TO PFKEY-RETURN-INDICATOR
GO TO P-100-PFKEYS-RETURN.
```

- For COBOL with mainline option 3:

```
.
. (remarks that describe processing)
.
IF PFK1
.
. (processing logic goes here)
.
MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
GO TO P-100-PFKEYS-RETURN.
```

- For PL/I with mainline option 1:

```
.
. (remarks that describe processing)
.
IF PFKEY_INDICATOR = '1'
THEN DO;
.
. (processing logic goes here)
.
PFKEY_RETURN_INDICATOR = 'R';
GOTO P_100_PFKEYS_RETURN;
END;
ELSE;
```

- For PL/I with mainline option 3:

```
.
. (remarks that describe processing)
.
IF PFKEY_INDICATOR = '1'
THEN DO;
.
. (processing logic goes here)
.
CONTROL_INDICATOR = DO_TRANSFER_LIT;
GOTO P_100_PFKEYS_RETURN;
END;
ELSE;
```

Non-PF-Key Logic in PF-Key Section

PF-key logic occurs as the first section executed on input processing. This section is normally used for PF-key processing. Therefore, it is named P-100-PFKEYS, and it is requested via the PFKEY parameter on the Create/Update Screen Definition screen. Sometimes, however, it is desirable to include COPY code (procedural logic) that is unrelated to PF-key processing in section P-100-PFKEYS.

For example, rather than following the normal processing sequence on input (D-100, E-100, X-100, H-100, etc.), you can control the sequence via procedural code. Logically, you do this in the first input section executed, P-100-PFKEYS. First, specify a character string in the PFKEY parameter. Then include the code under the name hhPFKccc, where ccc is the character string specified in the PFKEY parameter. As with normal PF-key processing, use the control indicator to indicate the action to be taken after the procedural logic is executed, and branch to P-100-PFKEYS-RETURN at the conclusion of the section.

HOLD Processing

Define two PF keys for use in holding and resuming processing. Use the HOLD PF-key logic—which defines explicitly the screen to gain control while the current screen is held—for each program that can be held. Use the RESUME PF-key logic for each program to which the HOLD logic passes control, either directly or indirectly (for example, in each program from which HOLD processing can be resumed).

PF-key logic is implemented through the PFKEYS parameter, as stated earlier.

Include the following steps in the logic for HOLD PF-key processing:

- Move 'D' to the reserved variable HOLD-AREA-TYPE
- Execute the program section, L-100-HOLD-SAVE
- Move the screen ID of the program gaining control to the reserved variable, NEXT-PROGRAM-NAME-ID

The logic for RESUME PF-key processing must include the following steps:

- Move 'D' to the reserved variable HOLD-AREA-TYPE
- Execute the program section, K-200-HOLD-RESUME

Example

The following example includes processing for two screens called MR50 and MR61. PF-key processing in MR50 provides for holding the conversation and transferring to menu screen MR10. Through a series of processes, the menu eventually passes control to MR61, which includes PF-key processing to return to the held screen, MR50.

- For COBOL:

(Hold in MR50)

```
IF PFKEY10-22
MOVE 'D' TO HOLD-AREA-TYPE
PERFORM L-100-HOLD-SAVE
MOVE 'MR10' TO NEXT-PROGRAM-NAME-ID
GO TO P-100-PFKEYS-RETURN.
.
.
.
```

(Resume in MR61)

```
IF PFKEY11-23
MOVE 'D' TO HOLD-AREA-TYPE
PERFORM K-200-HOLD-RESUME
GO TO P-100-PFKEYS-RETURN.
```

- For PL/I:

(Hold in MR50)

```
IF PFKEY_INDICATOR = 10 | PFKEY_INDICATOR = 22
THEN DO;
HOLD_AREA_TYPE = 'D';
CALL L_100_HOLD_SAVE;
NEXT_PROGRAM_NAME_ID = 'MR10';
GOTO P_100_PFKEYS_RETURN;
END;
```

.
.
.

(Resume in MR61)

```
IF PFKEY_INDICATOR = 11 | PFKEY_INDICATOR = 23
THEN DO;
HOLD_AREA_TYPE = 'D';
CALL K_200_HOLD_RESUME;
GOTO P_100_PFKEYS_RETURN;
END;
```

HELP Processing

Define two PF keys for use in requesting help text and returning from the HELP display screen. Use the HELP-request logic—which defines explicitly the messages to be displayed for the screen—for each program that can request screen-level text. Generally, the logic is unique for each screen since the displayed messages vary. Use the return logic for each program that displays help text.

You implement PF-key logic through the PFKEYS parameter on the Create/Update Screen Definition screen.

Include the following steps in the logic for HELP-request processing:

- Move 'P' to the reserved variable HOLD-AREA-TYPE.
- Execute the program section L-100-HOLD-SAVE.
- Move the key(s) of the message(s) to be displayed to the reserved variable, HELP-MSG-NAME(n), where n indexes the HELP-MSG-NAME array in the transfer work area. Note that a different area in the transfer work area can be used, but HELP-MSG-NAME is usually appropriate since it must be there for field-level HELP processing.
- Move a value to the HELP-MSG-COUNT field that corresponds to the number of MOVE statements coded (for example, the number of key values moved) for the previous step.
- Initialize the HELP-CURR-MSG-COUNT field to 1.
- Move the screen ID of the HELP-text screen to the reserved variable, NEXT-PROGRAM-NAME-ID.

Example

- For COBOL (two messages displayed):

```
IF PFKEY4-16
  MOVE 'P' TO HOLD-AREA-TYPE
  PERFORM L-100-HOLD-SAVE
  MOVE 'SSMR60A' TO HELP-MSG-NAME(1)
  MOVE 'SSMR60B' TO HELP-MSG-NAME(2)
  MOVE 2 TO HELP-MSG-COUNT (2 = highest helpkey value)
  MOVE 1 TO HELP-CURR-MSG-COUNT
  MOVE HELP-SCREEN-PGM-ID TO NEXT-PROGRAM-NAME-ID.
```


- For PL/I (three messages displayed):

```
IF PFKEY_INDICATOR = 4 | PFKEY_INDICATOR = 16
  THEN DO;
    HOLD_AREA_TYPE = 'P';
    CALL L_100_HOLD_SAVE;
    HELP_MSG_NAME(1) = 'MR10A';
    HELP_MSG_NAME(2) = 'MR10B';
    HELP_MSG_NAME(3) = 'MR10C';
    HELP_MSG_COUNT=3; (3 = highest helpkey value)
    HELP_CURR_MSG_COUNT = 1;
    NEXT_PROGRAM_NAME_ID = HELP_SCREEN_PGM_ID;
  END;
```

The return processing, coded in the HELP screens, must include the following steps:

- Move 'P' to the reserved variable, HOLD-AREA-TYPE. Be sure to use the P, rather than the D that is used to return from HOLD processing; otherwise, a screen-flow error results.
- Execute the program section, K-200-HOLD-RESUME.

Example

- For LCOBOL:

```
IF PFKEY5-17
  MOVE 'P' TO HOLD-AREA-TYPE
  PERFORM K-200-HOLD-RESUME.
```

- For PL/I:

```
IF PFKEY_INDICATOR = 5 | PFKEY_INDICATOR = 17
  THEN DO;
    HOLD_AREA_TYPE = 'P';
    CALL K_200_HOLD_RESUME;
  END;
```

SELECT Fields

SELECT fields are a common technique used in CA Telon to enter data that controls the flow of processing. Specifically, you can design screen fields defined as SELECT. By entering data in a SELECT field, the application user can execute the processing that corresponds to that field. Furthermore, since by definition an application user can choose only one SELECT field per screen, this request to process one field precludes processing for any other SELECT fields.

You do not have to use SELECT fields (or PF keys, discussed above) to control the processing flow. You can add special code to the consistency edit section (X-100-CONSIS-EDITS) using the CONSIS field on the Create/Update Screen Definition screen to analyze inputs and transfer control in lieu of using the SELECT fields. However, CA Telon automatically includes edits to verify that only one select field has been requested, which saves edit logic. If no select option is entered where one is required, the error message (TPO-ERRMSG1) is set to the ERROR-MESSAGE-NOHIT reserved field. If more than one select option is entered, the error message is set to the ERROR-MESSAGE-MULTHIT reserved field.

To define a SELECT field, use the select-field definition character during screen design. For further information, refer to the *Design Facility Reference*.

When you code the non-procedural statements for the screen, you can control the related processing using the following fields defined on the Update Select Field screen.

- NEXTPGM—Used to provide the name of the program to which control passes if this field is selected
- SCONSIS—Used to define code to be included to process the field, and corresponding selection option, when the field is selected
- INEDIT—Used to indicate whether the generated input edits are executed for the screen when the select option is chosen
- INDBIO—Used to indicate whether the generated file update processing is executed for the screen when the select option is chosen
- SELKEY—Used, during SEGLOOP processing, to identify the line for which the application user wishes to invoke further processing (discussed previously under *List Processing, SEGLOOP*)

These fields are referenced in the discussion that follows. The rest of this subsection covers the following considerations related to select-field processing:

- Determining where to pass control—How the CA Telon application determines where to branch to, depending on the SELECT field entered
- Processing with SELECT logic—Information related to the coding and use of user-defined (vs. system-generated) logic to process a select option
- SELECT fields within a loop—Considerations related to using a SELECT field to choose one of several options displayed using loop (for example, SEGLOOP) processing
- Altering a select option—How to control which select options display at any given time and suppress the functioning of a particular option

Determining Where to Pass Control

A major consideration when a CA Telon program recognizes that a SELECT field has been entered is to determine where to pass control. You define the screen to which control passes for the SELECT field using either the NEXTPGM field on the Update Select Field, Create/Update Screen Definition, or the SCONSIS field on Update Select Fields as follows (also discussed earlier under "Screen to Screen"):

- When all options access the same screen, identify that screen using the NEXTPGM keyword field on the Create/Update Screen Definition screen.
- When each option accesses a different screen, but the different screens are always consistent by option, identify the screens using the NEXTPGM field on the Update Select Field screen for each of the SELECT fields.
- When an option can cause a transfer to more than one screen, use custom code to determine which screen is required and to pass control to it. Identify the copy member that includes the custom code to be used for each SELECT field, using the SCONSIS field of the Update Select Field screen for that SELECT field. Considerations when using the SCONSIS logic are discussed next.

Processing Considerations with SELECT Logic

When a screen contains at least one SELECT field, the generated input edits (E-100-INPUT-EDITS logic), generated file update processing (H-100-INPUT-TERM logic), and custom code CONSID edits (X-100-CONSID-EDITS logic) for the screen are not executed automatically. If the generated edit and update processing is required, use the INEDIT and INDBIO parameters, respectively, on the Update Select Fields screen to request their execution automatically. To execute consistency logic (or, alternatively, to execute the generated edit and update processing), you can execute the corresponding code explicitly from the SCONSIS logic.

An SCONSIS copy member can include any normal processing functions (edits, I/Os, etc.). It is executed only when the associated SELECT field is filled in and no other SELECT field is entered.

Often, the special processing for the select option is quite extensive. In this case, you should use a structured approach for ease of maintenance: break the processing into multiple functional procedures or sections, and include them using the SECTION field on the Create/Update Screen Definition screen; then, in the SCONSIS code, include only the control logic to perform the independent sections/procedures.

The processing order for the field selected follows that described below :

- If editing or mapping is specified on the Update Select Field screen for the SELECT field (for example, using the FLDTYPE, DBNAME, or RANGE fields, the generated code for this field only is executed. If no errors are detected, the application continues with SELECT field processing.
- If you request the standard input-edit processing (INEDIT=Y), the standard FLDTYPE and FORMAT editing and formatting for INPUT and OUTIN fields executes. If an error is detected, the standard edit error message is returned to the screen, and the application code waits for further activity on the part of the application user. If no errors are detected, the application continues with SELECT field processing.
- If you want to include tailored edit logic in the SCONSIS code, you can parallel the processing used by the standard edit. If an error is detected, include the following logic in your processing:
 1. Move an appropriate error message to the reserved field, TPO-ERRMSG1.
 2. If you are using mainline option 1, move an 'E' to the CONSYS-INPUT-ERRORS reserved field to indicate that an error occurred. If you are using mainline option 3, move DO-WRITE-LIT to the reserved field, CONTROL-INDICATOR, to indicate that an error occurred. If you are using mainline option 2, either action is valid.
 3. Highlight and position the cursor to the field involved in the error, by moving the ERROR-ATTR reserved field, to the attribute byte for that field.
 4. Skip to the end of the selection logic (J-100-SELECT-RETURN).

The first eight bytes of the root-segment key contain the LTERM name from the IOPCB. The code below illustrates this processing:

- For COBOL with mainline option 1:

```
.  
  . (edit logic)  
.  
IF...  
  MOVE 'message' TO TPO-ERRMSG1  
  MOVE 'E' TO CONSYS-INPUT-ERRORS  
  MOVE ERROR-ATTR TO TPO-fldname-ATTR  
  GO TO J-100-SELECT-RETURN.
```

- For COBOL with mainline option 3:

```
.
. (edit logic)
.
IF...
MOVE 'message' TO TPO-ERRMSG1
MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
MOVE ERROR-ATTR TO TPO-fldname-ATTR
GO TO J-100-SELECT-RETURN.
```

- For PL/I with mainline option 1:

```
.
. (edit logic)
.
IF... THEN DO;
TPO_ERRMSG1 = 'message';
CONSID_INPUT_ERRORS = 'E';
TPO_fldname_ATTR = ERROR_ATTR;
GO TO J_100_SELECT_RETURN;
END;
```

- For PL/I with mainline option 3:

```
.
. (edit logic)
.
IF... THEN DO;
TPO_ERRMSG1 = 'message';
CONTROL_INDICATOR = DO_WRITE_LIT;
TPO_fldname_ATTR = ERROR_ATTR;
GO TO J_100_SELECT_RETURN;
END;
```

For a description of the reserved fields used above, (TPO-ERRMSG1, CONSID-INPUT-ERRORS, CONTROL-INDICATOR, TPO-fldname-ATTR, and ERROR-ATTR), see the *Design Facility Reference*.

When you are ready to transfer control to another screen, set the value of the NEXT-PROGRAM-NAME-ID reserved field, as illustrated below (screen-id, below, is a field containing the ID of the screen to which control passes, or the screen ID itself):

- For COBOL:

```
MOVE 'screen-id' TO NEXT-PROGRAM-NAME-ID.
```

- For PL/I:

```
NEXT_PROGRAM_NAME_ID = 'screen-id'
```

This needs to be done only if the NEXTPGM parameter (discussed earlier) is not specified.

If you request the standard file-update processing (INDBIO=Y), the standard generated file-create or -update I/O is performed, unless it is skipped due to an error during either field edit processing or select consistency (SCONSIS) processing.

Flow with List Screens

A common select processing technique is to display a series of items, using CA Telon's looping (for example, SEGLOOP) facility, and to use a SELECT field to indicate which item to process. To do this, you must define a SELECT field that accepts the line number of the item to be processed. At run time, the application user enters the line number of the desired item in the SELECT field. CA Telon, in turn, passes the required information associated with that item to the output logic of another screen, used to process only the item selected.

The following examples show a panel image with SELECT fields and the necessary entries of the Update Select Field screen.

[illegible]

```

XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME DISPLAY USAGE SELECT LINE 19 COL 18 LTH 001

GENERAL: NEXTPGM PLVD SCON SIS      HELPM SG
*      INEDIT (Y/N) INDBIO (Y/N)
*      SELKEY FROM XFER-SAVE-KEY
*      TO XFER-EMPL-ID

MAPPING: DBNAME1
*      OF
*      DBNAME2
*      OF
*      INIT

EDIT:    FLDTYPE PARM LIST EXTENSION
*      SPEC (FORMAT/CONVERT/VALUES/RANGE)
*
*

ATTR:    ATTRPRO ATTRINT EACOLOR EAHIGH EVALID
FMEXIT  _ _  FMCNTL=MFS (Y/N)

```

```

XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME DELETE USAGE SELECT LINE 19 COL 49 LTH 001

GENERAL: NEXTPGM PLVZ SCON SIS      HELPM SG
*      INEDIT (Y/N) INDBIO (Y/N)
*      SELKEY FROM XFER-SAVE-KEY
*      TO XFER-EMPL-ID

MAPPING: DBNAME1
*      OF
*      DBNAME2
*      OF
*      INIT

EDIT:    FLDTYPE PARM LIST EXTENSION
*      SPEC (FORMAT/CONVERT/VALUES/RANGE)
*
*

ATTR:    ATTRPRO ATTRINT EACOLOR EAHIGH EVALID
FMEXIT  _ _  FMCNTL=MFS (Y/N)

```

```

XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME UPTSEQ USAGE SELECT LINE 20 COL 18 LTH 001
GENERAL: NEXTPGM PLVA SCON SIS      HELPM SG
*      INEDIT (Y/N) INDBIO (Y/N)
*      SELKEY FROM XFER-SAVE-KEY
*      TO XFER-EMPL-ID

MAPPING: DBNAME1
*      OF
*      DBNAME2
*      OF
*      INIT

EDIT:   FLDTYPE PARM LIST EXTENSION
*      SPEC (FORMAT/CONVERT/VALUES/RANGE)
*
*
ATTR:   ATTRPRO ATTRINT EACOLOR EAHIGH EVALID
FMEXIT  _ _  FMCNTL=MFS (Y/N)

```

```

XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME STARTBR USAGE SELECT LINE 20 COL 49 LTH 006
GENERAL: NEXTPGM PCVL SCON SIS STARTBR HELPM SG STARTBR

*      INEDIT (Y/N) INDBIO (Y/N)
*      SELKEY FROM
*      TO

MAPPING: DBNAME1 WK-START-BROWSE-KEY
*      OF
*      DBNAME2
*      OF
*      INIT

EDIT:   FLDTYPE PARM LIST EXTENSION
*      SPEC (FORMAT/CONVERT/VALUES/RANGE)
*
*
ATTR:   ATTRPRO ATTRINT EACOLOR EAHIGH EVALID
FMEXIT  _ _  FMCNTL=MFS (Y/N)

```

HELP/HOLD Processing

CA Telon includes an online HOLD facility and HELP facility. With the HOLD facility, you hold the screen you are working with by pressing the PF Key defined as HOLD. This saves the screen and displays the TDF Main menu allowing you to perform any TDF functions you desire. You return to the held screen by pressing the PF key defined as END HOLD.

The HELP facility provides general information about the screen you are viewing when you press the PF key defined for HELP. If you need specific information for a particular field on the screen, you enter a question mark (?) in the first position of that field. You return in either case by pressing the PF key defined as END or END HOLD.

More information on these facilities follows, beginning with the HOLD facility.

Note: There are some differences in the HELP and HOLD facilities between the IMS/DC and CICS environments. These differences are described later in this chapter, under "HOLD Requirements."

HOLD Requirements

To hold the screen, you simply press an application-defined PF key. Similarly, when you want to resume processing of the held screen, you press another application-defined PF key from the current screen.

For CICS, HOLD processing is accomplished by saving the SPA-AREA in temporary storage, as item 1 of the HOLD-AREA-KEY queue. This HOLD-AREA-KEY consists of:

- HOLD-AREA-LTERM—The four-character terminal ID
- HOLD-AREA-APPLID—The first three characters of the program name, whose format is installation dependent
- HOLD-AREA-TYPE—A one-character field that is set to 'D' for HOLD and 'P' for HELP. For IMS/DC, HOLD processing is accomplished by saving the SPA-AREA in a HOLD database, defined in the screen definition. Characteristics of the HOLD database are described later in this subsection.

Note that to use the HELP facility for a screen, you must have implemented the HOLD facility's variable storage requirements, described below.

Requirements for implementing the HOLD facility, described below, include PF-key definition, screen definition, database definition (for IMS), and variable storage definition.

PF Key Definition

Define two PF keys for use in holding and resuming processing. Use the HOLD PF-key logic—which defines explicitly the screen to which control passes when the current screen is held—for each program that can be held. Use the RESUME PF-key logic for each program to which the HOLD logic passes control, either directly or indirectly (for example, in each program from which HOLD processing can be resumed).

PF-key logic is implemented through the PFKEYS parameter on the Create/Update Screen Definition screen. See the subject earlier titled "PF Keys" for more information.

The logic for HOLD PF-key processing must include the following steps:

- Move 'D' to the reserved variable HOLD-AREA-TYPE
For more information, refer to the *Design Facility Reference*.
- Execute the program section, L-100-HOLD-SAVE
- Move the screen ID of the program gaining control to the reserved variable, NEXT-PROGRAM-NAME-ID

The logic for RESUME PF-key processing must include the following steps:

- Move 'D' to the reserved variable HOLD-AREA-TYPE
- Execute the program section, K-200-HOLD-RESUME

See the "PF Keys" subsection, under "In HOLD Processing," for an example of coding for HOLD processing.

Screen Definition Screen Requirements

For any screen having the capability to either hold the current screen or resume processing of a screen held previously, code HOLD=Y on the corresponding Create/Update Screen Definition screen.

IMS/DC Database Definition

For IMS/DC, you define the HOLD database in the screen definition using the Update DBMS Type and the Update Database Segment screens. The screen definition structure can vary (in a way similar to the WORKSPA database), but the following requirements apply:

- The root must be a 16-byte segment with a nine-byte key, made up of two components:
 - The first eight bytes must be either the IOPCB-LTERM or the IOPCB-USER-ID.
 - The last byte must be a D when used for the HOLD function or a P when used for the HELP function.
- Each additional segment must be in a vertical path and unkeyed. (Segments are always read and written using a path call.)
- Each SEGMENT statement must have a usage of HOLD specified.
- The total dependent segment size (for example, the sum of the lengths of all segments except for the root) must equal the value defined in the SPASIZE or WKSPASZ parameters specified on Update IMS/DC Driver Environment screen.

For further information on the SPASIZE and WKSPASZ parameters, refer to the *Design Facility Reference*.

CA recommends that your HOLD database structure be a 16-byte root followed by several 2,000-byte or 4,000-byte segments. If desired, the segment size can be equal to the spase, but program performance is often impacted when segment sizes exceed 8K. For nonconversational processing, the structure below the 16-byte root should match the WORKSPA structure.

A sample HOLD database definition follows.

Note: The first eight bytes of the root-segment key contain the LTERM name from the IOPCB.

```
DATABAS  DBNAME=HLDDBD01,KEYLEN=9,PCBNAME=HOLD
SEGMENT   HOLD,DBSEG=HOLDROOT,SEGKEY=IOPCB-LTERM,KEYLEN=9,
           IMSKEY=HLDRTKEY,SEGLTH=16
SEGMENT   HOLD,DBSEG=XFERA,PARENT=HOLDROOT,SEGLTH=2000
SEGMENT   HOLD,DBSEG=XFERB,PARENT=XFERA,SEGLTH=2000
```

Defining a DB2 Table

The following information describes how to define a DB2 table as an IMS HOLD database:

```
*****
* DCLGEN TABLE(TELON.HOLD) *
* LIBRARY(CSAP.DB2.DCLLIB(HOLD)) *
* QUOTE *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS *
*****

EXEC SQL DECLARE TELON.HOLD TABLE
( HOLD_USERID CHAR(8) NOT NULL,
  HOLD_TYPE CHAR(1) NOT NULL,
  HOLD_LEVEL SMALLINT NOT NULL,
  HOLD_PGM_ID CHAR(4) NOT NULL,
  HOLD_DATA VARCHAR(4030) NOT NULL
) END-EXEC.

*****
* COBOL DECLARATION FOR TABLE TELON.HOLD *
*****

01 DCLHOLD.
10 HOLD-USERID PIC X(8).
10 HOLD-TYPE PIC X(1).
10 HOLD-LEVEL PIC S9(4) USAGE COMP.
10 HOLD-PGM-ID PIC X(4).
10 HOLD-DATA.
49 HOLD-DATA-LEN PIC S9(4) USAGE COMP.
49 HOLD-DATA-TEXT PIC X(4030).

*****
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5 *
*****
```

Variable Storage Definition

In the application work area (hhWKAREA) and transfer work area (XFERWKA) for the application, include fields for use by the generated HOLD Logic. These fields are listed below.

Note: The hhWKAREA fields are always included, so no special coding is necessary.

In the application work area, include the following fields:

- ERROR-MESSAGE-HOLD, used to define the message moved to TPO-ERRMSG1 upon return from the HOLD processing.
- ERROR-MESSAGE-HOLD-ISRT, used to define the message moved to TPO-ERRMSG1 when a HOLD request fails because a HOLD is already in process. (HOLDS cannot be nested within an application.)
- ERROR-MESSAGE-RESUME, used to define the message moved to TPO-ERRMSG1 when a RESUME fails because no HOLD request was in process.

In the transfer work area, include the XFER-HOLD-INDICATOR, a 1-byte character field (defined as containing a space), used to indicate that a program has been resumed following a hold.

HELP Requirements

CA Telon's HELP facility allows you to obtain online instructions for filling out screens or screen fields. This is accomplished by the following:

1. Setting up HELP data fields in the transfer work area (for CICS, usually a table of keys of records stored on a VSAM file)
2. Holding the current screen using the HOLD facility
3. Transferring to a HELP screen that processes the HELP data in the transfer work area

Help text associated with an entire screen is known as *screen-level help text*; that associated with a particular field on a screen is known as *field-level help text*.

You can design and code CA Telon screens to present HELP data to the application user. No restriction is placed on the HELP format, so you can easily implement a single- or multiple-tiered approach to help text presentation. However, you should provide two types of information for field-level help text:

- A general description of the field and its use on the screen
- The editing criteria for the field (for example, the valid format for data to be entered)

Note: Since help text must be presented from a user's perspective, a group that is closely associated with the end user (rather than EDP) should be responsible for maintaining help text.

As noted above, the HELP facility makes use of CA Telon's HOLD facility; therefore, to implement help text, the HOLD variables (described in the previous chapter) must be defined. Note that when the Create/Update Screen Definition indicates **HELP=Y** and the **FIELD HELPMMSG** parameters are specified, the standard HOLD procedural code is generated into the program automatically.

To request screen-level help text, press an application-defined PF key. To request field-level help text, enter a special character (usually a question mark) in the field. In either case, when you want to resume processing of the screen, you either press an application-defined PF key or select an option from the HELP-text screen.

For a given application, there are five functions used to implement the HELP facility. These functions (three of which parallel those for the HOLD facility) include HELP database requirements (for IMS/DC), screen definition requirements, variable storage requirements, HELP message definition, and PF-key definition. Each function is discussed separately below.

HELP database requirements (IMS/DC only)

The HELP database can have any structure since the display and maintenance screens are simply CA Telon programs. Therefore, as with any CA Telon program, the database used as output (in this case, the HELP database) must be defined in the calling program (for example, in the screen definition of the screen from which HELP can be requested). An exception to this is when the calling program transfers control to the help text display screen via an IMS/DC message switch. If the IMS/DC transfer technique to be used is in question, define the HELP database with a segment type of DUMMY in each screen definition.

Screen definition requirements

For any screen from which you can request screen- or field-level help text, code **HELP=Y** on the corresponding Create/Update Screen Definition screen.

Variable storage requirements

In the application work area (hhWKAREA) and transfer work area (XFERWKA) for the application, include fields for use by the generated HELP Logic. These fields are listed below.

Note: The hhWKAREA fields are always included, so no special coding is necessary.

In the application work area, include the following fields:

- ERROR-MESSAGE-HELP, used to define the message moved to TPO-ERRMSG1 upon return from a HELP request.
- HELP-FIELD-PGM-ID and HELP-SCREEN-PGM-ID, used to name the programs used, respectively, to display field-level and screen-level help text.

Note: The two programs can be identical; in this case, the application user gets the same screen (with a different message) whether the user requests screen- or field-level text.

- HELP-CHAR, used to define the one-byte character which the application user uses to request field-level help text for the screen. CA recommends use of a question mark (?).
- HELP-TABLE-LIMIT, used to define the maximum number of entries in the transfer work area's HELP message table (described below).

In the transfer work area, include the following fields:

- HELP-CURR-MSG-COUNT, which contains the number of the message (in the HELP-text-key table) being displayed currently on the HELP screen. This field is set automatically to 1 prior to control being passed to the HELP-FIELD-PGM-ID, but the programmer must set it (if needed) when screen-level HELP is requested.
- HELP-MSG-COUNT, which contains the number of message keys provided when each HELP request is made. As each field that has the HELP-CHAR as its first character is processed, the HELPMMSG key value is loaded into the message table and the HELP-MSG-COUNT is incremented.
- HELP-MSG-NAME, an array of keys to the HELP file or database. Define the maximum number of entries in this array using the HELP-TABLE-LIMIT field, described above.

HELP Message

Screen level

For the CICS environment, the help text is stored on a VSAM file. This file can have fixed or variable length records of any size, with any key length.

For CICS, you can create and maintain the help-text file in batch mode or online. Generally, a simple five to six screen CA Telon system is used for this purpose. The use of an online facility for maintenance is recommended to allow for timely updates to messages by the application end users. This approach places the responsibility for accurate and useful messages on the people who best understand the HELP informational requirements.

You should take the following precautions:

- Limit the responsibility for updating the messages to a minimum number of people (usually *supervisors* since the messages are generally available to all users and need to be somewhat standardized).
- Limit authority to delete messages to the application development staff rather than the end user. This is because the screens have been programmed to present specific messages when HELP is requested. Therefore, the application must be modified if a specific message is to be deleted.

For the IMS/DC environment, the help text is stored on a database. You can display and maintain it through standard CA Telon programs.

In IMS/DC, the help text database can have any structure, as long as maintenance and display screens are designed to support it. Each program requesting HELP, however, must pass the message keys to be displayed in the transfer work area in table HELP-MSG-NAME. Each key can be used by itself or in conjunction with other data passed in the transfer work area (such as the application header) to identify uniquely the message to be displayed to the application user.

Again, the help text display screen is a user-defined CA Telon program and can, therefore, be flexible in design. However, you should note the following programming considerations:

- Since the help text display screen is part of the application requesting help, a unique HELP display program can be generated for each application. Usually, a standard screen definition is created and only the application header (HEADER parameter on the Create/Update Screen Definition screen) is changed to generate the program.
- When the help text display screen returns to the program that transferred to it (by means of a standard CA Telon transfer technique, such as SELECT fields), two functions must be performed:
 1. Move 'P' to the reserved variable, HOLD-AREA-TYPE.
 2. Execute the program section, K-200-HOLD-RESUME.

The following example shows help text using PF-key processing.

- For COBOL:

```
IF PFKEY5-17
  MOVE 'P' TO HOLD-AREA-TYPE
  PERFORM K-200-HOLD-RESUME
  GO TO P-100-PFKEYS-RETURN.
```


■ For PL/I:

```
IF PFKEY_INDICATOR = 5 | PFKEY_INDICATOR =17
  THEN DO;
    HOLD_AREA_TYPE = 'P';
    CALL K_200_HOLD_RESUME;
    GO TO P_100_PFKEYS_RETURN;
  END;
```

Field-level text definition

To define the data (usually a record key) to be passed to the field-level HELP program when the HELP-CHAR is entered in the field, code the HELPMMSG parameter on the Update Select Fields screen. For example, if you code:

```
XXXXXX.PD UPDATE SELECT FIELD ***** *****
COMMAND ==>
FIELD NAME _____ USAGE SELECT_ LINE 03 COL 003 LTH 041

GENERAL: NEXTPGM _____ SCONSID _____ HELPMMSG MSG255 _____
*          INEDIT (Y/N) INDBIO _ (Y/N)
*          SELKEY FROM _____
*          TO _____

MAPPING: DBNAME1 _____
*          OF _____
*          DBNAME2 _____
*          OF _____
*          INIT _____

EDIT:      FLDTYPE _____ PARM LIST EXTENSION _
*          SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*
*
*

ATTR:      ATTRPRO _ ATTRINT _____ EACOLOR _____ EAHIGH _ EVALIDID _
          FMTEXTIT _ _ _ _ FMTONTL=MFS N (Y/N)
```

```

XXXXXX.PD UPDATE SELECT FIELD *****
COMMAND ==>
FIELD NAME _____ USAGE SELECT_ LINE 03 COL 003 LTH 041

GENERAL: NEXTPGM _____ SCONSI _____ HELPMMSG MSG102_____
*          INEDIT (Y/N) INDBIO _ (Y/N)
*          SELKEY FROM _____
*          TO _____

MAPPING: DBNAME1 _____
*          OF _____
*          DBNAME2 _____
*          OF _____
*          INIT _____

EDIT:      FLDTYPE _____ PARM LIST EXTENSION _____
*          SPEC _____ (FORMAT/CONVERT/VALUES/RANGE)
*          _____
*          _____
*          _____
*          _____

ATTR:      ATTRPRO _____ ATTRINT _____ EACOLOR _____ EAHIGH _____ EVALID _____
           FMTXIT _____ FMTCTL=MFS N (Y/N)

```

Then, when the application user places a question mark (for example, the HELP-CHAR) in both of these fields and presses Enter, the CA Telon program automatically:

- Puts the characters 'MSG255' in HELP-MSG-NAME(1)
- Puts the characters 'MSG102' in HELP-MSG-NAME(2)
- Moves 1 to HELP-CURR-MSG-COUNT
- Moves 2 to HELP-MSG-COUNT
- *Holds* the current screen
- Transfers to the HELP-FIELD-PGM-ID

PFKEY Use

Screen-level

Define two PF keys for use in requesting help text and returning from the HELP display screen. Use the HELP-request logic—which defines explicitly the messages to be displayed for the screen—for each program that can request screen-level text. Generally, the logic is unique for each screen, as the displayed messages vary. Use the return logic for each program that displays help text. PF-key logic is implemented through the PFKEYS parameter on the Create/Update Screen Definition screen. See the subsection "PF Keys," above, for more information.

For CICS, the logic for HELP-request processing must include the following steps:

- Move 'P' to the reserved variable, HOLD-AREA-TYPE.
- Execute the program section, L-100-HOLD-SAVE.
- Move the key(s) of the message(s) to be displayed to the reserved variable, HELP-MSG-NAME(n), where n indexes the HELP-MSG-NAME array in the transfer work area.

Note: A different area in the transfer work area can be used, but HELP-MSG-NAME is usually appropriate because it must be there for field-level HELP processing.

- Move a value to the HELP-MSG-COUNT field that corresponds to the number of MOVE statements coded (for example, the number of key values moved) for the previous step.
- Initialize the HELP-CURR-MSG-COUNT field to 1.
- Move the screen ID of the HELP-text screen to the reserved variable, by setting the LINEOPT parameter on the appropriate screen. This parameter generates line optimization logic in custom code. You can select one of the following values with the corresponding results:
 1. Use CA Telon line optimization subroutine
 2. Simulate subroutine optimization in custom code
 3. No special line optimization code is generated

OUTIFIL parameter

You can request the null fill of INPUT, OUTIN, and SELECT fields using the OUTIFIL parameter on the Create/Update Screen Definition screen. This eliminates the transfer over the TP line of a significant number of blank characters, especially on an add screen. One disCA, however, is that if the application user moves the cursor in a field using the cursor control keys and then enters data, that data is left shifted in that field when it is received at the CPU. You must make a trade-off between line efficiency on the one hand and the operational characteristics of the system on the other.

Note: CA Telon converts all nulls to spaces on input; therefore, the OUTIFIL specification is transparent to the application program itself.

If you omit the OUTIFIL parameter, then the fill default value specified at system installation time is used. For further information, refer to *Design Facility Reference*.

EOFKEY parameter

You can request that only data modified by the application user (only fields where the user keys in data) be transferred across the TP line to the CPU. This eliminates the transfer of a significant number of characters over the TP line, especially on error iterations. CA Telon merges the data transferred with that which it knows is on the screen to make it look like all the data in fields was transferred.

There is an operational impact, however, when using this option. Since IMS's MFS does not distinguish between a field that was not modified and one on which the application user pressed the EOF key on the first character, the application user cannot eliminate the field by simply pressing EOF. To eliminate a field, the application user must either blank out the field or enter at least one blank before pressing EOF. If you specify EOFKEY=**N** on the Create/Update Screen Definition screen, you are requesting that such line optimization be performed, understanding the application user restriction above. If you specify EOFKEY=**Y**, you request no such line optimization, but the application user does not have the above restriction.

If you omit the EOFKEY parameter, then the default value specified at system installation time is used.

Refresh considerations

All CA Telon IMS programs pass TP buffer fields from iteration to iteration for use in TP line optimization. This information passes either through the SPA, if IMS conversational processing is used, or through the WORKSPA, if non-conversational processing is used. You cannot eliminate this area, since CA Telon requires it. You can, however, impact the amount of data stored using the REFRESH parameter on the Create/Update Screen Definition screen. The correct use of the REFRESH parameter depends on a trade-off between the size of the SPA/WORKSPA area and user function.

REFRESH=Y

Using this option, all OUTPUT fields defined on the screen are carried across iterations. This is required if you implement either the HELP or HOLD functions in your application. Do not use this option if HELP/HOLD are not used. Note that more space is required in the SPA/WORKSPA area when you specify REFRESH=**Y**.

REFRESH=N

Using this option, only INPUT, OUTIN, and SELECT fields defined on the screen are carried across iterations. This saves space in the SPA/WORKSPA area, as less elements need be defined. However, the HELP/ HOLD functions cannot be used with this option.

REFRESH default

If you omit the REFRESH parameter, then the default value specified at system installation time is used.

Output field attributes

A somewhat related option is that of having the attributes of output fields carried in the output buffer. It does not affect TP line optimization, but does impact the size of the SPA/WORKSPA area. At system installation time, the option can be chosen so that attributes for all output fields are carried in the output buffer. This has the CA that the programmer can simply reference the attribute in procedural code without taking other action. It has the disCA that the attributes take up space in the SPA/WORKSPA area. If the installation option is not always to carry output attributes, this decision can be overridden on a field-by-field basis using the OUTATTR parameter on the Update Select Field screen. This is the preferred option, but does require you to perform two actions to modify the attribute for an output field: specify OUTATTR and write the custom code itself.

IMS/DC Report Processing

IMS/DC systems often transmit report information to hardcopy terminals. You can define such a report with CA Telon similar to the way you define a display screen. Define the report format (using only LITERAL and OUTPUT fields) and create a report definition.

Capabilities

The CA Telon IMS online reports enable you to:

- Create and transmit a report to an IMS-defined printer
- Generate single- or multiple-page reports
- Transmit reports to one or more IMS-defined printers
- Define a report similar to the way you define a screen

CA Telon IMS online reporting provides a:

- Report program structure that is similar to the output side of a CA Telon screen program. The main sections are:
 - A-100-OUTPUT-INIT (OINIT1 and OINIT2)
 - B-100-OUTPUT-EDITS (OCUST1, 2, and 3)
 - C-200-TERMIO-WRITE

- CA Telon report program that is called subroutine from a higher level program.

Uses

You can use these online reports:

- For interoffice mail
- To edit an error analysis report
- To perform abnormal error handling
- To create a customer information report
- To create refund checks

Differences between Screens and Reports

The major differences between using screens and reports in CA Telon are:

- A report definition includes a REPORT statement in place of a SCREEN statement.
- OUTPUT fields are the only variable fields that you can define in a report definition because no input from a hardcopy terminal is allowed.
- The report definition is not restricted to CRT terminal sizes. It can be any size that is valid for the printer terminal.
- A report program runs as a subroutine under a higher-level program (usually a screen program) rather than as a stand-alone transaction-driven program.

The rest of this subject describes the report definition and provides information on the report program structure, on controlling the destination of a report, and on the interface with the calling program.

Report Definition

A report definition is similar to a screen definition (as discussed above). CA Telon generates a REPORT statement (instead of a SCREEN or DRIVER statement) to indicate that the definition is for a report.

Required steps

The steps to define an IMS report are:

1. Paint the panel image
2. Define output fields using a panel definition
3. Specify the report definition panel
4. Create report custom code
5. Create file group parameters
6. Define the report environment

The parameters for the REPORT statement are similar to those for the SCREEN statement. The REPORT statement has one additional parameter, LINKWKA. LINKWKA is used to pass a user work area from the calling program to the print subroutine, allowing the subroutine access to the data used in the report. LINKWKA is discussed in more detail later in this chapter.

Valid generation statements

The only valid generation statements in a report definition are IMSPGM and IMSMFS. TSOPGM is not valid since there is no special TSO test version of a report program.

In this case, any TP call (inserted to the message queue for the printer terminal) is traced (a trace screen can be requested), but is not actually executed (that is, nothing is written). This allows the I/O area to be reviewed in testing for accuracy. To get a layout of the report, however, the program must be run under IMS.

In a report definition, the only valid parameters on the IMSPGM statement are: TRACE, GENPCBS, LNKCOPY, USGCOPY, PGMNAME, and MFSMOD.

A TPPCB statement must be included in the report definition, unless the XFER-PCB is used to define the terminal destination for the report. Report destination considerations are discussed later in this subject under "Controlling the Report Destination".

Paging

Paging cannot be requested for a report (that is, you cannot specify PAGE=Y on the SEGLOOP statement). If paging is necessary, the report program must be called once for each page. For more information, see "Calling Program Interface" later in this chapter.

Report definition example

In the following sample report, note that custom code is added (OINIT1=LTERMINT) to set the print terminal destination based on information in the transfer work area. Since the transfer work area contains all data passed to the report, the LINKWKA parameter is not specified on the REPORT statement. A TPPCB named PRINT is included to define the destination of the report (indicated by the PRINT=Y parameter). You can modify the destination since no LTERM is specified.

```
REPORT PRNT,HEADER=TR,XFERWKA=TRXFERWK,OINIT=LTERMINT
  TPPCB PRINT,EXPRESS=Y,PRINT=Y
*****
* EMPLOYEE DATA BASE *
*****
DATABAS DBDNAME=TRGDBD01,KEYLEN=16,PCBNAME=EMPLOYEE
SEGMENT BROWSE,DBSEG=TRGEMPL,IMSKEY=TRGEMKEY,KEYLEN=06,
PARENT=0,SEGKEY=EMPL-KEY
*****
* LITERALS *
*****
FIELD LITERAL,POS=(01,15),TEXT='T E L O N T R A I N I
N G S Y S T E M'
FIELD LITERAL,POS=(02,28),TEXT='EMPLOYEE LIST'
FIELD LITERAL,POS=(03,02),TEXT='PAGE'
FIELD LITERAL,POS=(06,02),TEXT='SEQ ID'
FIELD LITERAL,POS=(06,15),TEXT='NAME'
FIELD LITERAL,POS=(06,42),TEXT='SEQ ID'
FIELD LITERAL,POS=(06,55),TEXT='NAME'
FIELD LITERAL,POS=(06,82),TEXT='SEQ ID'
FIELD LITERAL,POS=(06,95),TEXT='NAME'
*****
* VARIABLES *
*****

DATE2 FIELD OUTPUT,POS=(01,73),LTH=008,FMTCTL=MFS
TIME FIELD OUTPUT,POS=(02,73),LTH=008,FMTCTL=MFS
PAGE0 FIELD OUTPUT,POS=(03,07),LTH=002,FLDTYPE=NONE
MORE FIELD OUTPUT,POS=(04,02),LTH=007,FLDTYPE=NONE
SEGL00P OUTPUT,INCR=(2,3,2,3,2,3,2,3,2,3,2,3,2),
LINECNT=SEQ,PAGE=N,CINCR=(40,40)
SEQ FIELD OUTPUT,POS=(09,02),LTH=002,FLDTYPE=NONE,PIC='Z9'
ID FIELD OUTPUT,POS=(09,07),LTH=006,DBDNAME=EMPL-KEY
NAME FIELD OUTPUT,POS=(09,15),LTH=020,DBDNAME=EMPL-NAME
CITY FIELD OUTPUT,POS=(10,02),LTH=024,DBDNAME=EMPL-CITY
STATE FIELD OUTPUT,POS=(10,30),LTH=002,DBDNAME=EMPL-STATE
SEGEN0
IMSPGM PGMNAME=TRPRINT,TRACE=N
IMSMFS
END
```

Report Program Structure

The structure of the report program is very similar to the output side of a screen program. The main difference is that the report program contains no cursor positioning logic. Parameters on the REPORT statement can be used to add custom code: OINIT1 and OINIT2 to add custom code to the A-100 sections; OCUST1, OCUST2, and OCUST3 to add loop-processing logic to the B-100 section; and PGMCUST to add further custom code to any section. In a report program, only three sections are executed from the main section:

- A-100, to perform output initialization including reading database segments and executing OINIT1 and OINIT2 code
- B-100, to perform output editing, including SEGLOOP processing
- C-200, to write a message to the terminal

Controlling the Report Destination

The destination (that is, the hard copy LTERM name) of a report can be defined either as a constant or as a variable.

Specify the destination of a report by indicating the teleprocessing PCB to be used. The report is routed to the hardcopy terminal whose logical terminal name (LTERM) is associated with that PCB. The characteristics of a PCB (including the LTERM associated with the PCB) are determined by the PSB used by the program.

In report processing, no PSB is generated for the program; instead, the report uses the PSB of the calling program. Therefore, the LTERM parameter on the TPPCB statement is not valid for a report definition.

As a constant

To define the report destination as a constant, a TPPCB must be defined and passed to the print subroutine (see "Calling Program Interface" later in this subject). This must be a non-modifiable TPPCB whose destination was defined upon generation of the PSB for the calling program. To indicate to the report which TPPCB is used to define the print destination (other TPPCBs can be defined for other functions, such as handling catastrophic error messages), specify PRINT=Y on the correct TPPCB statement.

As a variable

To define the report destination as a variable, the report program must use a modifiable TPPCB. This program can be either the XFER PCB, which is passed as a standard in most CA Telon IMS programs, or any modifiable TPPCB defined by using a TPPCB statement. If a TPPCB statement with PRINT=Y is included, then that PCB is used as the print destination; otherwise, the XFER PCB is used.

In either case, you determine the print destination by adding procedural code using the OINIT1 or OINIT2 exits to move the destination LTERM to the CA Telon field PRINT-LTERM-NAME. If this 8-byte field is equal to either spaces or the destination already set in the PCB, no CHNG call is executed. Otherwise, a CHNG call to the value in PRINT-LTERM-NAME is executed. In either case, the message is then inserted to the correct PCB. If you want the message to end in a PURG call, then include code within the OINIT1 and OINIT2 code mentioned above to move a "Y" to the CA Telon variable PRINT-PURGE- INDICATOR.

Calling Program Interface

The report program is executed as a subroutine, usually from a CA Telon screen program. Since the code is generated, it has specific interface requirements which dictate the parameters passed. The contents of the parameter list must be as described below.

Note: In PL/I programs, the print subroutine must be defined in the program (using the WKAREA parameter on the SCREEN statement). For example:

```
DCL PRNTSUB EXTERNAL ENTRY;
```

1. LINK-WORK-AREA—The area defined using the LINKWKA parameter. If LINKWKA is not coded, a DUMMY area must be passed. For COBOL, when a dummy area must be passed, you can call it PRNTSUB-AREA. For PL/I, this parameter must always be coded:

```
ADDR(PRINTSUB_AREA).
```

2. SPA-XFER-WORK-AREA—The SPA work area. For COBOL, this must be coded:

```
SPA-XFER-WORK-AREA.
```

For PL/I, this must be coded:

```
ADDR(SPA_XFER_WORK_AREA).
```

Note: SPA-AREA cannot be passed due to the variance in the length of the SPA-HEADERS.

3. IO-PCB—The IO PCB for the report definition.
4. XFER-PCB—The PCB for the transfer work area.
5. TPPCBs—The teleprocessing PCBs in the report definition, if any.
6. DBPCBs—The database PCBs in the report definition, if any.

Note: The order of the TPPCB and DATABAS statements in the report definition determines the order of the TPPCBs and the DBPCBs.

A sample call when no work area is defined using the LINKWKA parameter and a single TP PCB is used for the report destination follows.

- For COBOL:

```
CALL 'TRIMPRNT' using PRINTSUB-AREA  
SPA-XFER-WORK-AREA  
IO-PCB  
XFER-PCB  
PRINT-PCB  
EMPLOYEE-PCB.
```

- For PL/I:
CALL TRIMPRNT (ADDR(PRINTSUB_AREA),
ADDR(SPA_XFER_WORK_AREA),
IO_PCB
XFER_PCB
PRINT_PCB
EMPLOYEE_PCB.

Line Optimization Considerations

CA Telon provides routines to minimize the number of characters transferred back and forth over the TP line. Since telecommunication line costs form a major part of the costs of an application using a remote network, this can reduce your costs substantially. This optimization takes the following forms:

1. For OUTPUT fields (protected fields), all trailing blanks are converted to nulls, which are not transmitted over the line.
2. For INPUT, OUTIN, and SELECT fields (unprotected fields), trailing blanks are optionally converted to nulls (see SCREEN statement, OUTIFIL parameter, below). Null characters in such a field can impact the application user, if he/she uses the cursor shift instead of the spacebar to move the cursor.
3. For INPUT, OUTIN, and SELECT fields, the Modify Data Tag (MDT) can be set off to eliminate the transfer of all fields into the computer, except for those modified by the application user. This is probably the preferred mode of operation. However, a problem can arise because the IMS MFS does not distinguish between an unmodified field and a field where the application user pressed the EOF key on the first byte of the field. (See SCREEN statement, EOFKEY parameter, below.)

LINEOPT Parameter

For CICS or IMS, you control use of line optimization by setting the LINEOPT parameter on the appropriate screen. This parameter generates line optimization logic in custom code. You can select one of the following values with the corresponding results:

1. Use the CA Telon line optimization subroutine
2. Simulate subroutine optimization in generated code
3. No special line optimization code is generated

OUTIFIL Parameter

You can request the null fill of INPUT, OUTIN, and SELECT fields using the OUTIFIL parameter of the SCREEN statement. This eliminates the transfer over the TP line of a significant number of blank characters, especially on an add screen. One disCA, however, is that if the application user moves the cursor in a field using the cursor control keys and then enters data, that data is left shifted within that field when it is received at the CPU. You must make a trade-off between line efficiency on the one hand and the operational characteristics of the system on the other.

Note: CA Telon converts all nulls to spaces on input; therefore, the OUTIFIL specification is transparent to the application program itself.

If you omit the OUTIFIL parameter, then the fill default value specified at system installation time is used. For further information, refer to the *Design Facility Reference*.

EOFKEY Parameter

You can request that only data modified by the application user (only fields where he/she keys in data) be transferred across the TP line to the CPU. This eliminates transferring a significant number of characters over the TP line, especially on error iterations. CA Telon merges the data transferred with the data that it knows is on the screen, thus making it appear as if all data in the fields was transferred.

There is an operational impact, however, when using this option. Since IMS MFS does not distinguish between an unmodified field and one on which the application user pressed the EOF key on the first character, the application user cannot eliminate the field by simply pressing EOF.

To eliminate a field, the application user must either blank out the field or enter at least one blank before pressing EOF. If you specify EOFKEY=N on the SCREEN statement, you are requesting that such line optimization be performed, understanding the application user restriction above. If you specify EOFKEY=Y, you request no such line optimization, but the application user does not have the above restriction. If you omit the EOFKEY parameter, the default value specified at system installation time is used.

Refresh Considerations

All CA Telon IMS programs pass TP buffer fields from iteration to iteration for use in TP line optimization. This information passes either through the SPA, if IMS conversational processing is used, or through the WORKSPA if non-conversational processing is used. You cannot eliminate this area, because CA Telon requires it. However, you can impact the amount of data stored using the REFRESH parameter on the SCREEN statement. The correct use of the REFRESH parameter depends on a trade-off between size of the SPA/WORKSPA area and user function.

REFRESH=Y

Using this option, all OUTPUT fields defined on the screen are carried across iterations. This is required if you implement either the HELP or HOLD functions in your application. Do not use this option if HELP/HOLD are not used. Note that more space is required in the SPA/WORKSPA area when you specify REFRESH=Y.

REFRESH=N

Using this option, only INPUT, OUTIN, and SELECT fields defined on the screen are carried across iterations. This saves space in the SPA/WORKSPA area, as less elements need be defined. However, the HELP/ HOLD functions cannot be used with this option.

REFRESH default

If you omit the REFRESH parameter, then the default value specified at system installation time is used.

Output Field Attributes

At system installation time, you can place all output field attributes into the output buffer. This does not affect TP line optimization, but does impact the size of the SPA/WORKSPA area. The CA to this is that the programmer can simply reference the attribute in procedural code without taking other action. The disCA is that the attributes take up space in the SPA/WORKSPA area.

You can override this decision on a field-by-field basis using the FIELD statement OUTATTR parameter. This is the preferred option, but it does require you to perform two actions to modify the attribute for an output field: specify OUTATTR and write the custom code itself.

Chapter 10: Prototyping Facility

The Prototyping Facility, option 6 on the TDF Main menu, is a tool to prototype panel images and panel definitions. It is designed to include the application user in interactive sessions of defining screen functions and immediately reviewing those functions with simple scenarios. These sequences of screen samples simulate application functions. This facility is used for prototyping with or without data mapping.

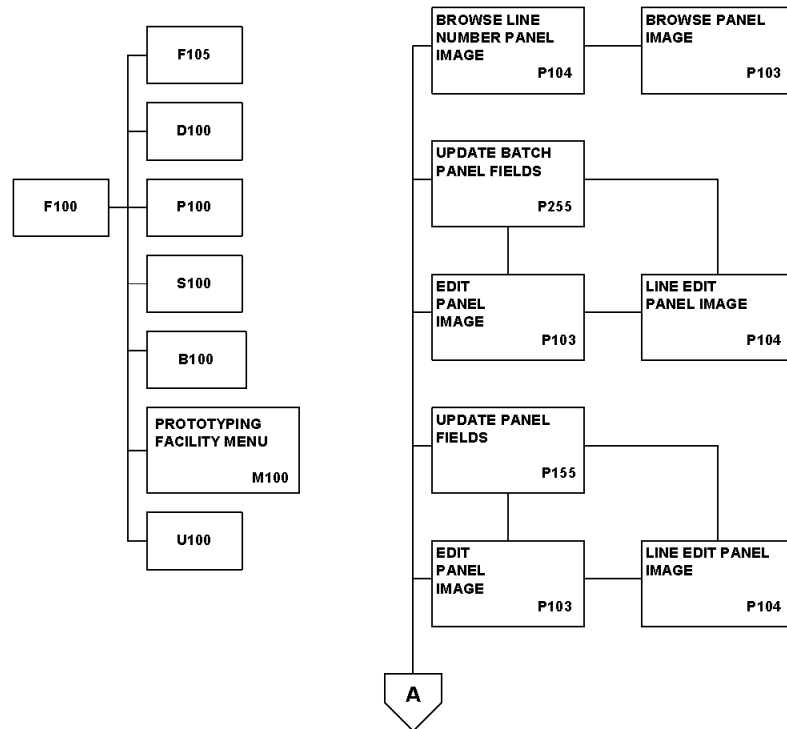
The goal of prototyping is to illustrate to the application user how the panels (screens) might appear during system execution and how they might interact with other panels to perform user functions. The Prototyping Facility requires design input of a panel image, with or without a panel definition. When a panel definition exists, the field information specified is used to enter data, pass it to other panels, and display data. If special edits have been set in the panel definition, these are also activated in the prototype.

Prototyping sessions are simple to create and perform. To simulate application execution, you need to perform the following steps:

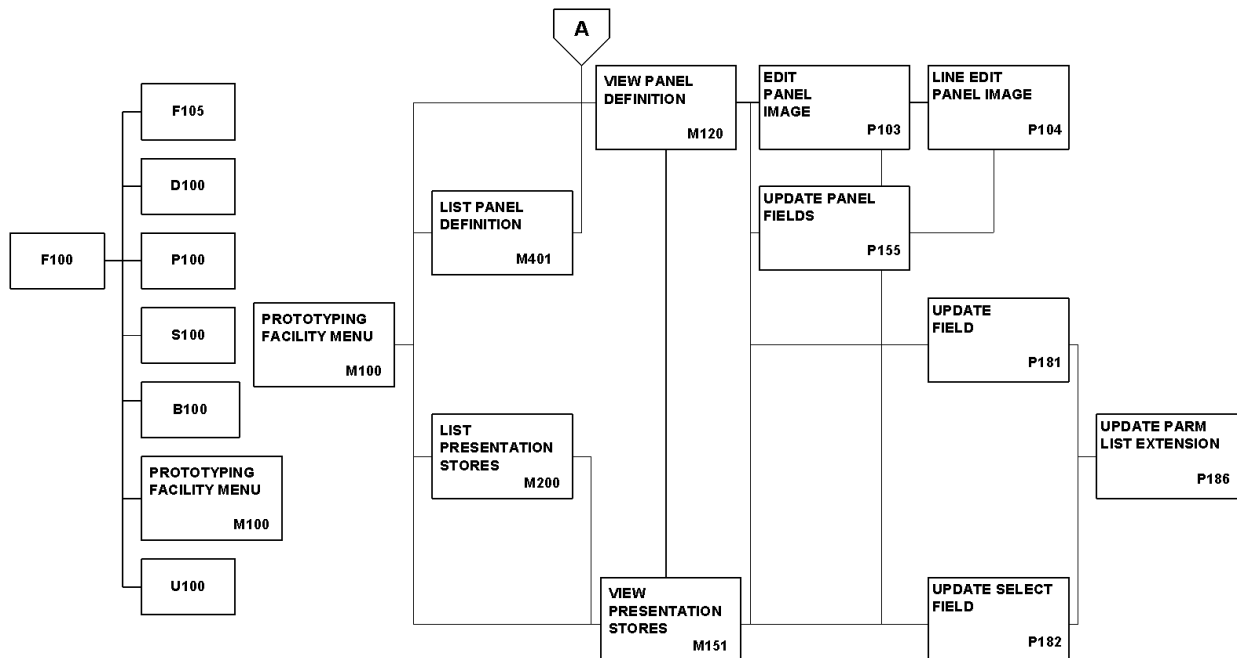
1. Enter data
2. Optionally edit the data
3. Transfer control from panel to panel

CA Telon automates this process for you using the Prototyping Facility. This chapter discusses how to use the Prototyping Facility and takes you step-by-step through four prototyping sessions. The following diagram shows the Prototyping Facility screen flow. For complete information on each screen shown in this flow diagram, refer to the *Design Facility Reference*.

Prototyping Facility hierarchical chart



Prototyping Facility hierarchical chart (continued)



Prototyping without Data Mapping

Prototyping without data mapping allows you to create application scenarios very quickly. The only design component required for this simulation is a panel image. This level presents screen designs to the application user for review. The user can comment on screen fields and functions and on screen-to-screen flow. You can enter data in the screens and move from screen to screen to simulate proposed work flows.

At this level, data is *not* stored for later display. This means that the application user does not see the data transfer that can be achieved at the next prototyping level.

This level can be combined with the next level, prototyping with data mapping. That is, panel images that have no panel definitions can be combined in the same scenario with panel images that have panel definitions. The data entered and displayed for the panel definitions is ignored for the panel images that have no panel definitions.

The main difference between levels 1 and 2 is that in level 2 information is stored and automatically mapped to other panels. This is accomplished through the use of a presentation store.

Prototyping with Data Mapping

Prototyping with data mapping allows you to make relatively realistic scenarios. The goal of this level is similar to the last—to present screen designs to the application user for review, both for general screen content and for screen-to-screen flow. This level uses the additional information contained in the panel definition to communicate more information to the reviewers, making the scenarios more realistic.

Since this level uses the full panel definition (as opposed to only the panel image in the previous level), the prototype can be made to look as if it were executing with sample databases or files. The element mapping information provided in the panel definition is used to enter data, edit it, and transfer it to later screens in the scenario. The data used is stored in a form called a presentation store.

Presentation stores

A presentation store is an unstructured collection of sample data values used in prototyping. You can enter values directly in the store using the CA Telon Editor or let CA Telon put in the store automatically as the result of data entered in application screens. Presentation stores are used both to transfer data from screen to screen for more realistic screen flows and to present screen designs displaying sample data. CA Telon automatically creates an active presentation store when you use prototyping with data mapping. In addition, presentation stores can be created, saved, and updated under application user control.

Field editing

You can use the following FLDTYPES and SPCLEDITS at the Prototyping Facility level:

```
DATE REQ=Y|N|C
JULIAN
FLOAT CONVERT
NUMERIC VALUES
FULLNUM RANGE
FULLCAR FORMAT
DOLLAR
LALPHA MAPOUT parameter
NBALPHA INIT parameter
STATE
```

Field editing at the Prototyping Facility level is similar to field editing of a compiled CA Telon program. FLDEDITS and special edits are specified for a field in the same manner as for compiled programs.

Flow control

You can control panel-to-panel flow by using the following:

- NEXTPGM parameter:
 - On the SELECT field statement
 - In the screen definition defined for the panel (lowest priority)
- NEXT-PROGRAM-NAME-ID as the DBNAME for INPUT or OUTIN fields

If you enter the ID of the next panel to be shown in the SELECT field's NEXTPGM parameter in the panel definition and select that field by entering a non-blank character, the Prototyping Facility transfers control to that panel ID after all edits have been satisfied.

If you have a valid ID in the NEXTPGM parameter of the screen definition defined for the panel, control flows to that ID if all edits are satisfied, and you have not used any of the two other methods of flow control (NEXTPGM parameter in the screen definition is the lowest priority). If you enter the DBNAME parameter of an OUTIN or INPUT field as NEXT-PROGRAM-NAME-ID and enter the ID of the next panel into that field, control flows to that panel after all edits have been satisfied.

Flow control is covered in greater detail elsewhere in this guide.

Prototyping Facility Menu

The Prototyping Facility menu displays the capabilities and features you can use to perform panel prototyping. If you know the name of the Header and ID of the panel on which you want to begin prototyping, you can begin your prototyping session from this screen. For further information, see "Initiating a Scenario from a List Screen" later in this chapter.

```

PROTOTYPING FACILITY MENU *****
COMMAND ==> _____

FUNCTION: VI VI-VIEW LI-LIST

ITEM AP AP-APPLIC PS-PRESENTATION STORE

START NAME:
  HEADER TR____
  ID MENU____
  PS NAME_____

ENTER DEFAULTS FOR INITIALIZING SCREEN FIELDS:
PRESENTATION: + < | > (OUTIN INPUT SELECT OUTPUT CHARACTERS)
              U U U B (OUTIFIL OVERRIDE - B-BLANK,U-UNDERLINE,N-NULL)
              %       (UNDEFINED STORE CHARACTER)
              Y       (CONVERT INPUT TO UPPER CASE - Y/N)
              2       (PROTOTYPE LEVEL - 1-WITHOUT DATA, 2-WITH DATA)
              Y       (FIELD EDIT AND FLOW EXECUTION - Y/N)

COMMAND POS: ____ (ROW - COLUMN OR
                NAME OF FIELD)
N INTENSITY (N-NORMAL,B-BLANK,H-HIGH)

```

To start prototyping without data mapping screens from the Prototyping Facility menu, enter:

Command	Parameter	Description
FUNCTION	VI	To request the view function.
ITEM	AP	To identify that an application scenario is to be viewed.
HEADER	name	To identify the application.
ID	name	To identify the panel to start the scenario.
PS NAME	name	To identify the presentation store to be retrieved for use in the scenario. This is required only when you are using Prototyping with mapping and have saved a group of field values to be used in the scenario. If a presentation store is specified, the active presentation store is initialized with the data in the presentation store retrieved. If you do not specify a presentation store when you start a new prototyping session, the active presentation store is empty. If you are continuing a previous session, the active presentation store contains the data previously entered.

Position of Command Field

COMMAND POS indicates the position of the command field on the screen. Optionally, you can change the COMMAND POS field. The first line allows you to enter the row and column in which you want to place the **COMMAND** field. The COMMAND field allows you to enter commands when you are prototyping. The COMMAND field defaults to the lower-right corner of the View Panel Definition screen. You can move the COMMAND field based upon screen coordinates (for example, column and row) or panel field name.

The second line allows you to enter a panel field name (such as ERRMSG1) from the existing panel definition. The panel field name specifies where you want to place the COMMAND field. This line overrides the line above it.

Note: You can use a field name ONLY when a panel definition already exists using the field name specified in the panel definition.

The third line allows you to control the display **INTENSITY** of the COMMAND field. Options for displaying the field are:

- **N**—*normal* intensity using the outin character selected (default)
- **B**—*blank* invisible to the viewer (but still accessible to enter commands)
- **H**—*high* intensity using the outin character selected

Initiating a Scenario from a List Screen

To start a scenario from a list of application IDs, request the list from the Prototyping menu by entering:

Command	Parameter	Description
FUNCTION	LI	To request the <i>list</i> function.
ITEM	AP	To identify that <i>application IDs</i> are to be listed.
HEADER	name	To identify the application.
ID	name	To identify the panel in the list. If this field is blank, the list starts with the first ID under the HEADER.

In response to this request, CA Telon presents the List Panel Definitions screen. To start a scenario from this list screen, enter:

- **V** beside the panel ID to start the scenario.
- A **presentation store name** in the **PR STORE** column to identify a store to be used in the scenario. This is optional.

When the view process is complete, the List screen regains control.

Prototype Screens

After you enter the necessary values on the Prototyping Facility menu, or the List Panel Definition screen, press Enter. CA Telon then presents you with the prototype screen you requested. A sample prototyped Employee Add screen follows.

As you can see in the example, all outin, select, and input fields were changed to underscores. CA Telon performed this conversion based on instructions given to the Prototyping Facility Main menu.

EMPLOYEE ADD SCREEN

ID

NAME

ADDRESS

PHONE

SEX

HOURLY RATE

HOURS PER WEEK

DATE OF EMPLOYMENT %%%%%%%%%

DATE OF BIRTH

123456

-

Note: The horizontal line in the lower-right corner of the screen is the PROTOTYPING FACILITY COMMAND field.

Command Field

The Prototyping Facility adds a command field to each panel displayed during prototyping. You can control the location of the command field or use the default position in the lower-right corner of the screen.

In the command field, you can enter:

- Flow control instructions
- Prototyping commands
- General TDF commands

Each of these possibilities is explained below.

Flow Control

Flow control transfers control from panel to panel. You define the first panel in a scenario on the Prototyping menu. Then you define each subsequent panel by entering its panel ID in the command field. If the length of the name entered equals the panel ID length, then the Prototyping Facility assumes the panel to be displayed is for the HEADER used for the last panel displayed.

If the length of the name entered equals the sum of the HEADER and the panel ID length, then the Prototyping Facility assumes the HEADER to be used at the front of the name. Any other length generates an error.

Prototyping Commands

Prototyping commands execute specific prototyping functions. All prototyping commands begin with a period. These commands are listed in the following table.

Command	PF Action
.CLR	Clears variable names from the active presentation store.
.CV	Clears values in the active presentation store, but does not clear the variable names.
.ERR errno	Requests an error iteration be processed, returning the error message contained in the DBNAME ERRMSG1 (with SUB=errno) in the active presentation store.
.LOD psname	Loads the presentation store named psname and makes it the active presentation store.
.MRG psname	Merges the presentation store named psname into the active presentation store.
.OI	Requests that INPUT fields on a panel be treated as OUTIN fields. This allows INPUT fields to be prefilled with values from the active presentation store.
.SAV psname	Saves the active presentation store for future load under the name psname.
.SUB subno	Sets the subscript for any array variables when used by non-list screens. It is ignored by fields within a SEGLOOP.
.UPD id	Transfers to update the panel definition ID. ID is optional. If it is not specified, transfer is for the Panel being viewed.
.UPI id	Transfers to update the Panel Image ID. ID is optional. If it is not specified, transfer is for the panel being viewed.
.VPS	Displays all variables in the active presentation store.

General TDF Commands

You enter General TDF Commands in the Prototyping Facility command field. Precede all TDF commands with a slash (/). The most common commands entered are shown in the following table.

Command	PF Action
/=n	To transfer to other options.
/=X	Prototyping execution and exit the TDF.
/END	End prototyping execution and return to the Prototyping menu.
/HELP	Present HELP for the Prototyping Facility, particularly on the use of the command field.
/HOLD	Hold execution and transfer to the TDF main menu to perform another function.
/SWAP	Swap to another function that is executing concurrently through use of the HOLD command.

Methods to Control Screen Flow

Controlling screen flow is an important part of prototyping. Proper screen flow adds realism and helps the potential users understand how the screens will work together to accomplish their intended business function.

CA Telon provides five methods for controlling screen flow. In order of precedence, they are:

1. Command field entry
2. .COMMAND-FLOW presentation store entries
3. NEXTPGM parameter for panels with SELECT fields
4. NEXT-PROGRAM-NAME-ID for panels without SELECT fields
5. NEXTPGM parameter in the SD defined for the panel

Command Field Entry

The first method is to enter the ID of the panel that you want to gain control on the command field. *Entries in the command field always take precedence over any other type of flow control.*

For example, to simulate and display the selection of an item from a menu that is being prototyped, do the following when the prototype menu is displayed:

1. To simulate application user action, key in the function request on the menu (this initializes application activity)
2. Enter the ID of the next panel you want presented in the command field (default command field position is the lower-right corner of the screen)

.COMMAND-FLOW

The second method is used primarily to create *canned* scenarios. After the Prototyping Facility detects a .LOD or .MRG command in the command field, it searches that presentation store for a .COMMAND-FLOW DBNAME. If found, the Prototyping Facility uses the ID value assigned to that DBNAME to execute flow to that panel ID. This method does not require data entry on the panel or the command field, and takes precedence over methods 3 and 4. Note that .COMMAND-FLOW and .COMMAND-INIT are preceded by a period (.) to prevent confusion with other DBNAMEs.

Note: NEXTPGM parameter of Select Fields in PD and NEXT-PROGRAM-NAME-ID require a panel definition and are mutually exclusive.

For example, if you use SELECT fields, Method 4 does not affect transfer.

NEXTPGM Parm of SELECT Fields in the PD

The third method requires you to define SELECT fields for the panel transferring control. One of the parameters associated with SELECT fields is NEXTPGM. NEXTPGM identifies the name of the next program to be transferred to that SELECT field if it has a non-blank character entered in it. If such fields are defined for a panel, entry of a non-blank character in a given SELECT field transfers control to the panel ID listed at the NEXTPGM for that field. (If you want to edit the panel, you must specify the INEDIT parameter as **Y** on the SELECT field and satisfy all edits before the transfer can take place.)

NEXT-PROGRAM-NAME-ID

The fourth method requires the use of an INPUT or OUTIN field defined with NEXT-PROGRAM-NAME-ID as the DBNAME. Data entered into the panel for this field can be edited (and particularly CONVERTed) to the name of the next program in the scenario. When you press Enter, if all edits have been satisfied, CA Telon passes control to the program ID found in NEXT-PROGRAM-NAME-ID. After the transfer, CA Telon deletes NEXT-PROGRAM-NAME-ID from the active presentation store.

NEXTPGM Parameter of Screen Definition

The fifth method for controlling screen flow uses the NEXTPGM parameter in a screen definition defined for the panel. After field edits have been satisfied, the panel flow proceeds to the panel named in the NEXTPGM parameter of the screen definition for the current panel. This is the lowest precedence flow control technique. The command field technique, .COMMAND-FLOW, SELECT fields, and NEXT-PROGRAM-NAME-ID always take precedence. When displaying a panel, CA Telon positions the cursor at the screen's first non-output field.

You can also set up canned scenarios using presentation stores with embedded .MRG commands. This is explained later in this subsection under "Creating Canned Scenarios."

Terminating a Scenario

To end prototyping, type the command **/END** in the command field or press the End PF key. This returns you to the Prototyping menu.

When one scenario is ended and you want to start another, simply enter the ID of the first panel definition of the new scenario in the command field. If you want to clear the active presentation store prior to starting the scenario, enter **CLR** on the command field, as explained under "Presentation Stores". If you want to load information from a presentation store saved earlier, enter **LOD psname** on the command field, as explained under "Presentation Stores."

Modifying Application Definitions

During prototyping, you can modify any panel image or panel definition presented by switching immediately to the appropriate edit screen.

- Enter **UPI** in the command field to request the Edit Panel Image screen.
- Enter **UPD** in the command field to request the Update Panel Fields screen. This allows you to update the **panel definition**.

After updating either of these screens, press the End PF key to return to the Prototyping Facility with the updated panel image or panel definition.

To examine field-level information in a panel definition, type **!** in the first byte of the field. In fact, you can enter **!** in up to 19 fields at once. The Prototyping Facility will transfer you to the FIELD update screen for each selected field.

Otherwise, you can enter the **UPD** command to access the entire panel definition on the Update Panel Fields screen. In the case of a SEGLOOP screen, you must enter the **!** in the first occurrence of the SEGLOOP fields.

While in FIELD update, you can change or delete any item. You can then CANCEL or SAVE, and press the End PF key to return to the Prototyping Facility.

Presentation Stores

This subject covers:

- Mapping from a presentation store
- Contents of an active presentation store
- Modifying and saving an active presentation store

Mapping from a Presentation Store

If you are using prototyping with data mapping, the data passed from panel to panel is stored in a temporary data pool in memory. This data pool is called the *active presentation store*. At the beginning of a TDF session, the pool is empty unless you retrieve a saved presentation store.

As a panel is presented and executed, the Prototyping Facility adds the following to the active presentation store for the specific field name/DBNAME:

- Any fields in the panel definition for INPUT, OUTIN, and SELECT fields
- Data entered from the panel being prototyped for INPUT, OUTIN, and SELECT fields
- Blanks to initialize a field if no data is entered

During the execution of a scenario, the Prototyping Facility selectively maps data from the active presentation store to the panel for display. For any OUTPUT or OUTIN fields defined on the panel, the data in the presentation store with a matching DBNAME in the panel definition is mapped to the field on the panel for display.

If the field has not already been defined to the presentation store (see "Modifying an Active Presentation Store," below, for the various ways a field can be identified to the store), the field is filled with the undefined store character, identified on the Prototyping menu. This usually indicates an error condition, as the panel field does not have data previously defined in the scenario. If no mapping was defined on output (no DBNAME specified), the field is filled with a constant to show its position and indicate that mapping was not defined. The constant is either the panel image variable character or the user profile fill character, as specified on the Prototyping menu. The lack of mapping can indicate an error condition such as a forgotten specification, or more likely, detail to be added in a later step.

When you use data mapping (for example, you specify DBNAME), any OF specification for that field in the panel definition is ignored. Same DBNAME is used on two fields, each with a different OF name, the Prototyping Facility uses the same active presentation store field.

Use the PROTOTYPE LEVEL parameter to control mapping. In PROTOTYPE LEVEL 1 (prototyping without data mapping) CA Telon uses only panel image information to build the screen. This way, no mapping can take place and neither data entry nor panel flow affect your active presentation store. Since no mapping can take place, the Prototyping Facility cannot perform flow control using input from the screen other than from the command field.

If your default language is COBOL, use dashes (-) as a separator character for DBNAME. If your default language is PL/I, use underscores (_) as a separator character. All DBNAME examples in this chapter use the COBOL separator character of a dash.

Contents of an Active Presentation Store

When using the Prototyping Facility with data mapping, field editing, and flow control, data is mapped in and out of the data pool called the active presentation store. This area includes both data names (for example, DBNAMEs for fields within a panel definition) and data values associated with those names. Initially, the area is empty. DBNAMEs are added as panel definitions are presented. A field can be added either during output processing (when fields from the panel are displayed) or input processing (when the application user presses Enter and data is mapped into the system).

During Output Processing, OUTPUT or OUTIN field values default to undefined presentation store character. The Prototyping Facility adds a field to the active presentation store when the following occurs:

- The field is defined as an OUTPUT or OUTIN field in the panel definition
- A DBNAME (mapping name) is defined for the field in the panel definition
- The field name (DBNAME) does not already exist in the active presentation store

During Input Processing, the INPUT, OUTIN, or SELECT field values are set according to the screen field value or OUTIFIL default. The Prototyping Facility adds a field to the active presentation store when the following occurs:

- The field is defined as an INPUT, OUTIN, or SELECT field in the panel definition
- A DBNAME (mapping name) is defined for the field in the panel definition
- The field name (DBNAME) does not already exist in the active presentation store

Note: List screens form a special case and are discussed later in this subsection.

If you use field editing for a given screen, then the active presentation store contains the data in the same format as seen on the panel. In other words, the internal format described in the *Design Facility Reference* does not exist in the presentation store. For example, DATE-edited fields have the screen representation (MM/DD/YY) or (MMDDYY) rather than the internal representation (YYMMDD) stored in the presentation store. The only exception to this is CONVERT. CONVERTed data is in database format in the presentation store.

For example, **CONVERT=(1,ADDS,2,DISP,3,UPDT,4,LIST)** has ADDS in the presentation store if you enter **1** on the screen or DISP in the presentation store if you enter **2** on the screen; and so on.

Modifying an Active Presentation Store

To request an update of the active presentation store while:

- Viewing a panel, enter **VPS** in the command field
- On the Prototyping menu, enter the **FUNCTION VI** and the **ITEM PS**, leaving the PS NAME blank

In response, you transfer to the CA Telon Presentation Store Editor (the View Presentation Store screen), editing the Active Presentation Store. It lists all current fields and their values in alphabetical order by DBNAME. You can use this technique to change data to conform to the field edit specified for any or all fields.

You can alter, delete, or enter field values from this screen. To alter a field, type over the value of the field or its length. To delete it, type **D** on the same line and press Enter. To enter a new field, key the new DBNAME and data value into an empty line or a used line or use **I** in the line command area to insert.

When inserting be certain to enter, if necessary:

1. The correct LTH (default is 30)
2. The correct SUB (default is 1)

If a field is too long to fit in the VALUE field on one line, you can continue the VALUE data into the next line. To do this, press Enter to continue the data to the next line, leaving the subsequent **DBNAME** and **LTH** fields blank. Then change the first LTH field to the desired value.

You can save up to 1,000 names in the store.

Press the End PF key. This ends the update and returns to the previous function, which is either the Prototyping Facility menu or the panel you were viewing.

Saving and Retrieving a Presentation Store

Ending in the above manner updates only the *active* presentation store. If you want to save the updated store under a name, you can perform the following:

1. Create a new presentation store from the active presentation store screen you are viewing.
2. Return to the panel you are prototyping and save it.

Creating a new Presentation Store from the Active Store

Type the command **CREATE psname** in the command field and identify the lines you want included in the new presentation store by enclosing them with the CC line command or the C(n) line command.

Saving the Presentation Store from the Panel You are Prototyping

Press the End PF key to return to that panel and enter **SAV psname** in the command field. This saves the entire active presentation store. If no presentation store exists by that name, it is created. If a presentation store does exist by that name, it is replaced.

If you saved a presentation store earlier, you can load it in one of three ways depending on your location within the Prototyping Facility. If you are:

- Executing a panel, type the command **LOD psname** in the command field
- On the Prototyping Facility menu, enter FUNCTION **VI**, ITEM **AP**, and PS NAME **name** (the name of the presentation store to load)
- On the List Panel Definitions screen, enter **V** on the line for the panel to be presented, and the **psname** in the PR STORE/RENAME field

In each of these cases, the presentation store loaded overlays the current active presentation store.

Presentation stores are independent of the panel definitions themselves. That is, you can modify the panel definition and the presentation stores still function. If you add a new field to a panel definition, however, and that field requires data in the presentation store, you must add the data in the store. Otherwise, CA Telon fills the field with the undefined store character when the panel is displayed.

Presentation Store Editor

CA Telon provides a useful Presentation Store Editor. The editor uses standard line commands, such as CC, RR, MM, I, and D, within the formatted context of a Presentation Store screen.

The Presentation Store Editor lets you:

- Edit multiple presentation stores at one time
- Copy or move lines between stores
- Create a new store while editing another

When editing one store, you can add another store to the edit session. To add another store to the edit process, type **EDIT psname** in the command line at the top of the screen. This is similar to editing multiple custom code members with the Custom Code Editor.

To create a new store, follow the instructions given previously.

You save presentation stores from the command line. To save the stores:

- Individually, type **SAVE psname** for each store
- As a group, type **SAVE ALL**

When you edit and save a presentation store that has multiple variables with the same name and subscript (SUB) number, the Prototyping Facility keeps only the last or lowest occurrence of the variable. For further information on the Presentation Store Editor, refer to the *Design Facility Reference*.

Canned Demonstrations

Saving a presentation store is useful for presenting *canned demonstrations*. To set up data and use it to present an application scenario, perform the following steps:

1. Execute each panel in the scenario to add the names of all variables to the active presentation store. You can enter data values for those variables by either using the Presentation Store Editor or keying data into INPUT and OUTIN fields on the screen.
2. Save the active presentation store contents by creating a presentation store from that data.
3. Load the presentation store by name at the appropriate point in the application scenario (usually at the beginning).

You can also use presentation stores to present different kinds of sample data for a single display screen. To set up sample data and use it with a display screen, perform the following steps:

1. Execute the panel (view it using the Prototyping Facility)
2. Modify the values in the active presentation store to create each set of sample data and save each set under a presentation store name
3. View the display screen with each presentation store saved previously

Listing Presentation Stores

To obtain an alphabetical list of all presentation stores stored within the Prototyping Facility, enter the following on the Prototyping menu:

Command	Parameter	Description
FUNCTION	VI	To request the list function
ITEM	PS	To identify that presentation stores are to be listed
PS NAME	name	To identify the presentation store name at which the list starts

If the PS NAME field is blank, the list begins with the first presentation store in the file. If you enter characters in the **PS NAME** field, the list starts with the first presentation store containing those characters or the next presentation store in the collating sequence.

The **HEADER** and **ID** fields are ignored.

Prototyping List Screens

The execution of list screens forms a special case under the Prototyping Facility. When a panel image exists without a panel definition during prototyping, only the initial group of fields making up the proposed SEGLOOP are defined and presented. After you define the panel definition, you can create the SEGLOOP data required to correctly expand the List screen for prototyping.

CA Telon has two types of list screens:

- **File lists**, where a segment is read or written from each group processed.
- **Table lists**, where each group is mapped from or to a row in an array. The Prototyping Facility treats both types the same. The data is mapped in and out of an array in the active presentation store. The Prototyping Facility automatically manages the array. On a list screen, the array index is incremented from one to the maximum number of groups on a screen. Paging is not handled.

Mapping out from the array occurs for any fields with usage OUTPUT or OUTIN. Mapping stops when all groups in the presentation store have been processed. To determine the first nonexistent group, the Prototyping Facility processes each group in sequence from the array in the active presentation store until no more entries exist in the array.

Mapping into the array occurs for any fields with usage INPUT or OUTIN. Mapping stops on the first blank group on the screen or when all fields are processed. A blank group is one in which all fields in the group on the panel are blank. CA Telon ignores any index value specified in the panel definition for a DBNAME within the SEGLOOP (required for mapping to or from an array when a program is generated).

The creation of variable entries in the active presentation store works a little differently for list screens. As described earlier in this chapter, entries for fields not within a SEGLOOP are created on output or input, depending on how the field is defined. This is also true of fields within a SEGLOOP. The only special issue is determining how many occurrences of each field to create.

During output processing for a list screen, no additional occurrences of fields within a SEGLOOP are added to the active presentation store. Thus, if no DBNAMEs exist in the active presentation store, none are created. However, if some fields within a SEGLOOP exist for an occurrence, the other fields within the SEGLOOP are added for that same occurrence. For example, if EMPL-ID and EMPL-NAME exist within a SEGLOOP and the active presentation store contains variables **EMPL-ID** with SUB=1 and **EMPL-ID** with SUB=2, the Prototyping Facility adds EMPL-NAME with **SUB=1** and EMPL-NAME with SUB=2.

During input processing for a List screen, occurrences for INPUT and OUTIN fields are created for each group processed. As described earlier, a group is not processed on input if all the fields in the group are blank.

Handling Arrays from Non-List Screens

If you enter data from a List screen, CA Telon creates an occurrence (a subscript entry) for a group in the active presentation store for each panel group entered. Sometimes, however, you want to enter table data from non-list screens. This usually occurs when your application uses list screens to display data, but enters data from non-list screens. Thus, you can have a data entry screen where only a single occurrence for a group of fields is to be entered into an array. In this case, you need to be able to specify which occurrence of the group to enter, that is, to specify which subscripted entry in the table to use.

Similarly, you can display a particular occurrence of a group from the array. For these cases, the `.SUB` command specifies the subscript (occurrence number) for the array. For example, if you enter **SUB 3** before a display screen, CA Telon subscripts all input array variables (DBNAMEs defined for an array) with 3. If you do not enter data for a subscripted entry, the Prototyping Facility displays the undefined store character for that entry. Remember, you must set **SUB n** before data entry.

After you set a subscript, CA Telon considers **all** variables entered on that screen to be subscripted. The Prototyping Facility has no way of distinguishing subscripted from unsubscripted variables. If this creates undesirable entries in the active presentation store, you must delete them yourself.

Subscripted and Unsubscripted Variables

To better understand how the Prototyping Facility treats variables, you must know that a variable is considered:

- **unsubscripted** if the `.SUB` command
 - Has *not been executed*
 - Has been executed with a *subscript of one* (`.SUB 1`)
- **subscripted** if the `.SUB` command
 - Has been executed with a *subscript greater than one*

If you want to use both subscripted and unsubscripted variables in prototyping, be aware that:

- The Prototyping Facility **cannot distinguish between subscripted and unsubscripted variables**, so it must consider them both the same.
- To do this, *the Prototyping Facility views unsubscripted variables as having a subscript of one.*
- If you only want some of the input fields to be subscripted, you *must manually alter the active presentation store*. To unscript any variable names, simply space out the SUB value in the store or change it to one.

- If some unsubscripted variables in the active presentation store are to be displayed within a SEGLOOP on a list screen, the list screen is displayed with one occurrence of the group of fields. This means that at any one occurrence, the list screen is filled with pieces of information that all have the same subscript. Subscripts can be used to mark different instances of the same type of information.
- When you create subscripted variables from a non-list screen, all the variables on the screen are considered as either subscripted or unsubscripted.

Subscripted Variable Display Rules

CA Telon uses the following rules to determine what happens on output to unsubscripted variables if a subscript has been set, that is, if **SUB n** is specified, with n greater than 1.

When mapping variables to the screen with a subscript set, each variable is processed first as subscripted and then as unsubscripted.

For example, if the subscript is 5 and the variable being processed is EMPL-NAME, the Prototyping Facility first searches the active presentation store for EMPL-NAME with SUB=5. If the entry is found, it is mapped to the screen. If it is not found, the Prototyping Facility searches the active presentation store for EMPL-NAME with SUB=1 (in effect, an unsubscripted variable name, since its subscript is 1). If the entry is found, it is mapped to the screen. If it is not found, it is created using the undefined store character and then mapped to the screen.

A subscript specified with a .SUB command remains in effect from panel to panel until you do one of the following:

- Reset it by removing it with the command **SUB 1**
- Load a new presentation store with the **LOD** command
- Return to the Prototyping menu

Simulating Error Processing

You can simulate error processing using the .ERR command. This reviews the general characteristics of an error response with application users. To set up error scenarios, perform the following:

1. Be sure that your error message field defined in the panel definition is properly named as ERRMSG1
2. Enter each error message you wish to use into the presentation store with a DBNAME of ERRMSG1 and a unique subscript number (SUB number)
3. Request the simulation of an error by entering **ERR subno** in the command field, where subno is the subscript number of the error message to be returned

In response, the Prototyping Facility will:

- Leave all data on the screen
- Return the requested error message
- Reposition the cursor to the first field on the screen

The error simulation *does not highlight fields in error* or position the cursor at the first field in error.

Customizing Error Messages

The *Design Facility Reference* describes the standard CA Telon data names for the error messages found in xxWKAREA.

You can customize the error message for the following errors:

ERROR-MESSAGE-HIGHLIGHT

Produced: when a field edit has failed

Default: *** ABOVE HIGHLIGHTED FIELDS IN ERROR***

ERROR-REQ-CHAR

Produced: when a required field is not entered

Default: *

ERROR-MESSAGE-NOHIT

Produced: when a panel with SELECT fields detects that **all** SELECT fields are blank

Default: *** NO OPTION SELECTED ***

ERROR-MESSAGE-MULTHIT

Produced: when a panel with SELECT fields detects that more than one SELECT field contains a non-blank character

Default: *** MORE THAN ONE OPTION SELECTED ***

If you want a more appropriate message (based on size or content), add these to your presentation store:

- Above error message name as a DBNAME
- VALUE of the error message
- LTH of the error message

The Prototyping Facility detects and uses your customized value; otherwise, CA Telon uses the default message.

Creating Canned Scenarios

As described earlier in this chapter, you can simulate screen flows by entering the panel ID of the next screen to gain control in the command field at each screen. In addition, the .MRG command can be executed to add any presentation stores containing additional data values to be displayed. In response to a .MRG pstorage command, CA Telon merges data values from the presentation store pstorage with those from the active presentation store, replacing any duplicate values from the active presentation store and adding any new values.

Such scenarios can also be set up in advance and executed automatically (canned) without the application user entering any panel IDs or Prototyping commands. Two special variables in the presentation store let you can a scenario: .COMMAND-FLOW and .COMMAND-INIT. These variables start with a period to avoid confusion with normal variables. The variable .COMMAND-FLOW does differ from flow processing discussed earlier in this chapter under "Controlling Screen Flow." Its processing is automatic and not dependent upon data entered into any panel.

Special CA Telon Field Names

You can place the following special CA Telon field names in the presentation store for display on output:

- ERRMSG1
- SYSMSG
- MORE
- PAGENO
- TRANCDE

You cannot specify these fields with FLDTYPE=NONE. Therefore, they are not mapped in the standard way. To display the fields on the screen, they must have a SUB (subscript) value of 999 in the active presentation store.

.COMMAND-FLOW identifies a panel ID that receives control as soon as this presentation store is loaded (for example, with a .LOD or .MRG command). Thus, if you enter **MRG PS1** in the command field and PS1 has variable .COMMAND-FLOW defined with a value of DISP, the panel DISP is processed for output immediately after merging the presentation store PS1 into the active store.

.COMMAND-INIT identifies initialization values for the command field. Thus, if the active presentation store or the presentation store just loaded has variable .COMMAND-INIT defined with a value of .MRG PS1, the command field for the panel displayed is initialized to .MRG PS1.

The combination of the two commands can define a canned scenario. The scenario is started by entering the **MRG** command. The loaded presentation store must have both .COMMAND-FLOW and .COMMAND-INIT defined. The .COMMAND-FLOW value defines the next panel ID to be displayed. The .COMMAND-INIT value defines the .MRG command for the next presentation store to be loaded, with its related .COMMAND-FLOW value to determine the next flow.

As an example, assume you must create a canned scenario that begins with panel MENU, passes control to panel DISP and then back to MENU. You set up two presentation stores (named PSMENU and PSDISP) with the following COMMAND values:

```
PSMENU
.COMMAND-FLOW MENU
.COMMAND-INIT .MRG PSDISP

PSDISP
.COMMAND-FLOW DISP
.COMMAND-INIT .MRG PSMENU
```

To prevent mapping on flow control processing, prefix a greater than sign (>) to the beginning of the .COMMAND-FLOW value.

To start the scenario, type in .LOD PSMENU. This loads presentation store PSMENU, transfers to the MENU panel, and initializes the command field with .MRG PSDISP.

When you press Enter without altering the command field, presentation store PSDISP is merged into the active presentation store, control passes to the DISP panel, and the command field is initialized to .MRG PSMENU. When you press Enter again, presentation store PSMENU is merged into the active presentation store, control is passed back to the MENU panel, and the command field is initialized back to .MRG PSDISP.

You can alter the canned scenario flow by modifying the presentation store name in the command field. For the example above, if you want the DISP panel to transfer control to itself (simulating another display request instead of return to MENU), change the .MRG PSMENU command to **MRG PSDISP** and press Enter. The PSDISP presentation store is merged into the active store and control passes to DISP (the value in variable .COMMAND-FLOW). You could affect the same flow by changing .MRG PSMENU in the command field to DISP, but then the PSDISP presentation store would not be merged into the active presentation store.

The Presentation Store Editor allows you to edit multiple presentation stores at once and save time. Copying the commands can save key strokes and seeing the COMMAND values of all stores at the same time can clarify their values as they relate to the whole scenario.

Simulating Edit/Flow Scenarios

The Prototyping Facility lets you simulate a combination of field editing and flow control.

- To specify field editing, use FLDTYPE and/or SPEC edit parameters
- To specify flow control, use either the **NEXTPGM** parameter of the SELECT fields or the NEXT-PROGRAM-NAME-ID as a DBNAME for INPUT or OUTIN fields

To define an editing function for a given field, select one of the allowable edits, listed earlier in this subsection, and add it to the FLDTYPE or SPEC edit parameters for the field you want to edit. The Prototyping Facility then tests data entered from the screen or data in the presentation store.

Note: In panels with SELECT fields, editing for any field occurs only if **INEDIT=Y** for the SELECTed field.

OUTPUT

On the OUTPUT side of processing a screen, the Prototyping Facility edits data from the active presentation store. Any error messages appear in the command field with the format:

```
!FE,XX where XX='OE' means a FLDTYPE has failed on OUTPUT
XX='FM' means a FORMAT has failed on OUTPUT
XX='CV' means a CONVERT has failed on OUTPUT
```

If an error occurs, use the following methods to correct data for editing:

- For OUTPUT fields, enter either **VPS** or **UPD**:
 - **VPS** transfers you to the active presentation store edit session to correct the erroneous data
 - **UPD** transfers control to the field list panel from which you can change the field characteristics
- For INPUT fields, either change the value on the panel field or enter either **UPD** or **!'**:
 - **UPD** transfers control to the field list panel from which the field characteristics can be changed
 - **!'** In the first byte of an INPUT/OUTIN/SELECT field, **!'** transfers you to the Update Panel Fields screen where you can change the field characteristics

Note: **!'** does not save input when control passes to the field update.

Error messages for edits on the INPUT side of processing appear in the ERRMSG1 field (required for prototype level 2).

Flow control can be included as previously described by use of the **NEXTPGM** parameter in the panel definition for SELECT fields (SELECT screens) or the **DBNAME** of **NEXT-PROGRAM-NAME-ID** for INPUT or OUTIN fields (non-SELECT screens).

Command field processing flow control always takes precedence over any other type of flow control processing.

Input Mapping Considerations

While viewing a panel, you can enter a new panel ID or prototyping command in the command field to transfer to a new panel or perform a prototyping function. Whether data in INPUT, OUTIN, and SELECT fields is mapped into the active presentation store and when it is mapped depends either on what command you specify or how you entered the panel ID. This can be important to your prototyping scenario.

For example, suppose you are on a simple data entry panel that enters one occurrence of data into an array. If you transfer to that panel without having entered the **.SUB** command, the subscript defaults to 1. If you then key data into the panel and enter **SUB 2** in the command field, the Prototyping Facility first maps the data into the active presentation store using a subscript of 1. Then it processes the **.SUB** command to set the subscript to 2. If you had assumed that the subscript was set to 2 prior to mapping, the data entered would have been mapped to the wrong occurrence in the array.

The following table identifies the command entries for which data is mapped into the active presentation store and when that mapping occurs.

Command	Prototyping Facility Action
id	Map input, transfer to next panel.
>id	No mapping, transfer to next panel.
.CLR .CV .LOD	Mapping is immaterial for these three commands, since the commands destroy the contents of the active presentation store.
.ERR	No mapping, fill in ERRMSG1 with appropriate message, redraw screen.
.MRG	Map input, merge presentation store, transfer to output.
.OI	No mapping, transfer to output.
.SAV	Map input, save presentation store, redraw screen.
.SUB	Map input, change subscript, redraw screen.
.UPI .UPD .VPS	Map input, go to appropriate function. Upon return from that function, transfer to output for the panel being viewed at the time of the command.

Redraw screen means control returns to the panel being viewed, without any change to the screen contents, except for field ERRMSG1 on the .ERR command.

Transfer to output means control passes to the output side of the current panel being viewed, and all data for OUTPUT and OUTIN fields (and INPUT and SELECT fields if the .OI command has been executed) is mapped from the active presentation store to the screen.

Suggested Prototyping Steps

When using the Prototyping Facility, you can use various techniques to make the prototyping session more effective. The following are some suggested steps for including prototyping in your design process:

1. Gather *initial information* about the business function. This can include a hierarchical breakdown of functions and a general description.
2. Gather *initial information about the data* to be used and organize it into logical groupings or entities. A general understanding of the data and its structure is more important at this stage than an exhaustive list of data items and an accurate definition of each item's characteristics.
3. Identify *CA Telon application circles*, groupings of screens to perform related functions. This subdivides a large project into more manageable units.
4. Draw *circle flowcharts* and paint *panel images* for proposed screen layouts. Initial layouts can be rough, and perhaps incomplete, but they should contain at least enough information to perform the business function. The focus here is on function—how the screens interact to perform effectively. However, you should use existing screen design standards, since this eases later steps and makes communication more effective.
5. Use *Prototyping without data mapping* to prototype these business function scenarios for the application user and the department. The purpose is to test ideas and prompt further thinking about the effective implementation of business function.
6. Continue gathering information about the data until you have an exhaustive list and can make an *accurate definition of the data*. Then add it to the data structure.
7. *Modify panel images* to add data elements and alter function.
8. *Add panel definition* information to the panel images. Include data mapping and any special attributes needed to present the business function.
9. Use *Prototyping with data mapping* to prototype business function scenarios for the application user and the data process. *Presentation stores* to set up data and make the prototype scenarios more realistic. Begin to direct attention to how the screens are formatted and how the information looks displayed on these screens.
10. Create *initial application databases*. Make these as accurate as current information allows, although they are work versions that can be modified later.

11. *Create Screen Definitions*, defining general characteristics and data access. Functions such as HELP, HOLD, and PF-key processing can be added at this stage if they help the application user understand the execution of the application. Some *custom programming* can be required at this point.
12. *Generate* programs and *test* them. Introduce real application test cases at this point to rigorously test the design.
13. Use the information gathered from the final prototype to *complete application definition and programming*. Proceed into final application testing.
14. Determine *target environment* characteristics and generate the application into that environment.
15. Perform *final system testing*, thoroughly checking out the environment and exercising the system extensively.

Displaying Screens with Sample Data

You can use presentation stores as sample data to illustrate various panel formats. To do this, perform the following steps:

1. **Create** a presentation store for each type of sample data to be evaluated
2. **Load** one of the stores with the first panel to view
3. **Transfer** to each additional panel under consideration, one after the other
4. **Load** the next presentation store and *repeat* the process. In this manner, you can use the same sample data to evaluate the design of several panels.

Realistic Scenarios

Several easy techniques can make a prototype scenario seem more realistic:

- If you have a series of screens in a scenario, *enter some of the data to be displayed on those screens*. To do this, enter data for those fields into a presentation store, save the presentation store, and then retrieve it prior to beginning the scenario.
- When creating a new presentation store, you can *have the Prototyping Facility determine what variable names are necessary and initialize the names in the active presentation store*. To do this, use the Prototyping Facility to pass control to each panel that is to be a part of the scenario for which the store is used. Then edit the active presentation store using the .VPS command. The active presentation store includes the names of all variables used on the panels processed, but their values are filled with the undefined store character. Simply key in the new values and save the active presentation store under the new presentation store name.

- Set the command field to be *nondisplay* (INTENSITY BLANK).
- Use the .OI command to *show sample input data without keying it in* each time. If you set up a presentation store with the sample values you want for INPUT fields, as well as OUTPUT and OUTIN fields, entering the **OI** command causes those values to be displayed.
- The Prototyping Facility does *not* handle special fields, such as **MORE**, **PAGE**, and **LINECNT**, automatically. If you want to *simulate the special fields*, you have to specify DBNAME values for them in the panel definition and also in the presentation store. For example, if the LINECNT field for a SEGLOOP is field **LINENO**, you can specify a DBNAME of LINE-COUNT for that field in the panel definition. Also, in setting up the array of values for the SEGLOOP in the presentation store, you would include **LINE-COUNT** values (for example, **LINE-COUNT** with (**SUB=1**) = 01, **LINE-COUNT** with (**SUB=2**) = 02).

Control Hints

Several techniques can assist you in controlling the prototyping process. Since CA Telon does not force a presentation store naming convention, you can name the stores to make them most useful to you.

Presentation stores represent sample data and are not necessarily tied to any panel or application **HEADER**, *but*:

- If a store **is** tied to a single panel, begin the store name with the **HEADER** and **ID** of the panel
- If a store is tied to a single application **HEADER**, begin the store name with the **HEADER**
- If several presentation stores are versions of the same kind of data, begin the store names with a common prefix

SELECT Fields are important to the application and users need to be able to recognize them quickly on a panel. Use the *SELECT paint character* (I) for display rather than the OUTIFIL character (specified on the Prototyping menu) to make the fields easier to recognize.

Sample Prototyping Sessions

This part of the chapter presents step-by-step instructions for prototyping.

You will see how to implement the techniques discussed in the previous subjects with the informative sessions presented here.

All the prototyping scenarios presented here use the same panel images so you can see the distinct requirements and results of the different types of prototyping.

The examples illustrate the following types of prototyping:

- Without data mapping
- With data mapping

Prototyping without Data Mapping

To begin prototyping without a data mapping scenario, first create the panel images of the application screens. For further information, see Chapter 5, "Creating a Panel Image."

This example assumes that the panel images have been created for an employee system.

EMPLOYEE MENU SCREEN

1. EMPLOYEE ADD
2. EMPLOYEE DISPLAY
3. EMPLOYEE LIST
4. EMPLOYEE UPDATE
5. EXIT

OPTION <
ID <<<<<

EMPLOYEE ADD SCREEN

ID	+++++
NAME	<<<<<<<<<<<<<<<<<<<<<<
ADDRESS	<<<<<<<<<<<<<<<<<<<<<< <<<<<<<<<<<<<<<<<<<<<< << <<<<<
PHONE	<<<<<<<<<<
SEX	<
HOURS PER WEEK	<<<<
DATE OF EMPLOYMENT	+++++
DATE OF BIRTH	<<<<<<

>>>>>>> EMPLOYEE DISPLAY SCREEN

EMPLOYEE ID >>>>>

[illegible][illegible]

>>>>>>>>>>>>>>>>>>>

>> >>>>> >>>>>>>>>

DATE OF BIRTH >>>>>>> SEX >>>

DATE OF EMPLOYMENT >>>>>>>

HOURS PER WEEK >>>>>

DEPARTMENT >>>

HOURLY RATE >>>>>>

SALARY >>>>>>>>>

DISPLAY EMPLOYEE ID ||||| UPDATE EMPLOYEE ID |||||

[illegible]

PFKEYS: 2-HOLD 3-END 4-ENDHOLD 6-MENU 7-PREV 8-NEXT

>>>>>>> EMPLOYEE LIST SCREEN PAGE >>

>>>>>>>>

SEQ ID	/	NAME	/	ADDRESS	SEQ ID	/	NAME	/	ADDRESS

>> >>>>>>

[illegible]

>>>>>>>>>>>>>>>>>>

[illegible]

DISPLAY SEQ# || UPDATE SEQ# ||

[illegible]

PFKEYS: 2-HOLD 3-END 4-ENDHOLD 7-BACK 8-FORWARD

[illegible]

After you create the panel images, access the Prototyping Facility. To do so, enter option **6** on the Main menu as seen in the following screen.

```
TELON DESIGN FACILITY MAIN MENU *****
COMMAND ==> _____

FUNCTION: 6_

1 -- USER PROFILE MAINTENANCE
2 -- DATA ADMINISTRATION
3 -- PANEL SPECIFICATION
4 -- ONLINE PROGRAM DEFINITION MENU
5 -- BATCH PROGRAM DEFINITION
6 -- PROTOTYPING FACILITY
U -- UTILITIES
X -- EXIT
```


Prototyping Facility Menu

CA Telon then transfers control to the Prototyping Facility menu as shown in the next screen.

```

PROTOTYPING FACILITY MENU *****
COMMAND ==> _____

FUNCTION: VI VI-VIEW LI-LIST

ITEM AP AP-APPLIC PS-PRESENTATION STORE

START NAME:
  HEADER TR____
  ID MENU____
  PS NAME _____

ENTER DEFAULTS FOR INITIALIZING SCREEN FIELDS:
PRESENTATION: + < | > (OUTIN INPUT SELECT OUTPUT CHARACTERS)
               U U U B (OUTIFIL OVERRIDE - B-BLANK,U-UNDERLINE,N-NULL)
               %      (UNDEFINED STORE CHARACTER)
               Y      (CONVERT INPUT TO UPPER CASE - Y/N)
               2      (PROTOTYPE LEVEL - 1-WITHOUT DATA, 2-WITH DATA)
               Y      (FIELD EDIT AND FLOW EXECUTION - Y/N)

COMMAND POS: _____ (ROW - COLUMN OR
                        _____ NAME OF FIELD)
N INTENSITY (N-NORMAL,B-BLANK,H-HIGH)
  
```

Using the following parameters on the Prototyping Facility menu, you can begin a prototyping scenario of your panel image screens.

You enter:

Command	Parameter	Description
FUNCTION	VI	For view
ITEM	AP	For application
HEADER	TR	For the header of your application
ID	MENU	For the name of the screen

Next, enter the **U** to establish underscores as the character you want displayed for input, outin, and select (variable) data. **B** remains the default for output (protected) data. These fill characters override the field usage characters, and appear as you present the application screens during prototyping. Finally, change the prototyping level from the default (2) to **1**, and press Enter.

Prototyped Panel Images

CA Telon then displays the Employee menu screen. Note that on this and all other screens in this prototype session, all input and output fields appear as underscores. Output fields are blank.

After reviewing this menu with the application user, you find that it appears to be correct. The application user then asks you to simulate data being entered. To do this you enter **1** for **Option** and an **ID** of **123456**.

The application user is pleased, and then asks you to transfer to the Add screen. Because the panel definition is not created yet, you cannot transfer by pressing Enter. To transfer, type the ID of the Employee Add screen (**ADDS**) on the command field, and then press Enter.

EMPLOYEE MENU SCREEN 1. EMPLOYEE ADD 2. EMPLOYEE DISPLAY 3. EMPLOYEE LIST 4. EMPLOYEE UPDATE 5. EXIT OPTION 1 ID 123456 ADDS---

CA Telon then displays the Add screen. The employee ID did not carry over to this screen because you are prototyping without data mapping. Optionally, you can re-enter the Employee ID to provide a more realistic simulation.

After reviewing this menu, the application user informs you that you forgot the Hourly Rate field. You can request the Panel Editor to make the addition. To do so, type **.UPI** on the command field (this indicates an update to the panel image). Then press Enter.

EMPLOYEE ADD SCREEN

ID

NAME

ADDRESS

PHONE

SEX

_

HOURS PER WEEK

DATE OF EMPLOYMENT

DATE OF BIRTH

.UPI

As requested, the Employee Add screen appears in PI update (edit) mode. All field usages are displayed, and those fields with information contained in the PD are protected.

Enter the hourly rate field information (see the Employee Add screen). Now you need to update this information in the panel definition. You do not have to return to the Prototyping Facility to do this. You can perform a swap edit from this PI.

As requested, your panel change is saved, and you return to the Add screen in the Prototyping Facility. Note that your field addition is now displayed on the PI, below.

After data is added to this screen, the normal system work flow transfers to the Display screen to confirm that the added data on this screen is successfully processed. To simulate this work flow, type **DISP** on the command field, and press Enter.

EMPLOYEE ADD SCREEN

ID _____
 NAME _____
 ADDRESS _____
 PHONE _____
 SEX _____
 HOURLY RATE _____
 HOURS PER WEEK _____
 DATE OF EMPLOYMENT _____
 DATE OF BIRTH _____

DISP _____

The Display screen appears. In a real situation, if you had successfully added data via the Add screen, the added data would be displayed below. Since you are using level one prototyping, data is not mapped.

The application user reviews this screen and is satisfied with its content. To simulate the work flow, type **LIST** on the command field, and press Enter.

[illegible]

CA Telon transfers you to the Employee List screen.

The application user reviews this screen and is satisfied with its content. To simulate work flow, enter **UPDT** on the command field, and press Enter.

[illegible]

The Employee Update screen appears.

EMPLOYEE ADD/UPDATE SCREEN

ID
 NAME
 DATE OF BIRTH
 SEX
 PHONE

 STREET
 CITY
 STATE

 DATE OF EMPLOYMENT
 DEPARTMENT
 HOURLY RATE
 HOURS PER WEEK

 -

 _____ ZIP _____

/END

This completes the level one prototyping session without data mapping. To end a prototyping session, you must use a TDF command, not a prototyping command. To end the session, either:

- Press PF3.
- Enter **/END** in the command field

Note: A slash (/) must precede a TDF command issued from the Prototyping Facility.

Prototyping with Data Mapping

You begin your prototyping session *with* data mapping from the Prototyping Facility menu.

```

PROTOTYPING FACILITY MENU *****
COMMAND ==> _____

FUNCTION: VI VI-VIEW LI-LIST

ITEM AP AP-APPLIC PS-PRESENTATION STORE

START NAME:
HEADER TR____
ID MENU____
PS NAME _____

ENTER DEFAULTS FOR INITIALIZING SCREEN FIELDS:
PRESENTATION: + < | > (OUTIN INPUT SELECT OUTPUT CHARACTERS)
                U U U B (OUTIFIL OVERRIDE - B-BLANK,U-UNDERLINE,N-NULL)
                %      (UNDEFINED STORE CHARACTER)
                Y      (CONVERT INPUT TO UPPER CASE - Y/N)
                2      (PROTOTYPE LEVEL - 1-WITHOUT DATA, 2-WITH DATA)
                Y      (FIELD EDIT AND FLOW EXECUTION - Y/N)

COMMAND POS: _____ (ROW - COLUMN OR
                        _____ NAME OF FIELD)
N INTENSITY (N-NORMAL,B-BLANK,H-HIGH)

```

Using the following parameters on the Prototyping Facility menu, you can begin a prototyping session of your application screens. Enter:

Command	Parameter	Description
FUNCTION	VI	For view
ITEM	AP	For application
HEADER	TR	Header identifying your application
ID	MENU	Name identifying your first panel for simulation

Next, enter the **Us** to establish underscores as the fill character for INPUT, OUTIN, and SELECT type data. Enter **B** to establish a blank as the fill character for OUTPUT data.

Finally, make sure that the prototyping level is **2**, and press Enter.

The menu screen appears.

The application user indicates that he wants a few spaces above **1. ADD EMPLOYEE** deleted. To make this modification, enter the **UPI** command on the Prototyping Facility command field, and press Enter. The period tells CA Telon that this is a Prototyping Facility command.

```

EMPLOYEE MENU SCREEN

      1. ADD EMPLOYEE
      2. DISPLAY EMPLOYEE
      3. LIST EMPLOYEES
      4. UPDATE EMPLOYEE
      5. EXIT

OPTION _
ID _____

.UPI_____
    
```

Once you receive this screen in update mode, press PF10 to enter line edit mode.

Enter the **DD** commands to delete the extra spaces above **ADD EMPLOYEE**, as shown on 000003 through 000005. Press Enter.

```

COMMAND ==> SCROLL ==> CSR

000001      EMPLOYEE MENU SCREEN
000002
000003
000004
000005
000006
000007      1. ADD EMPLOYEE
000008
000009      2. DISPLAY EMPLOYEE
000010
000011      3. LIST EMPLOYEES
000012
000013      4. UPDATE EMPLOYEE
000014
000015      5. EXIT
000016
000017
000018      OPTION <
=PR0T>      ID <<<<<<
=PR0T>
000021
000022
    
```

LINE Commands

The standard line commands listed below can be entered in the command area on the left side of each line.

Command	Description
D	Delete line
I	Insert line
R	Repeat line
C	Copy line
M	Move line
A	After
B	Before
O	Overlay
)	Shift right
(	Shift left
X	Exclude line
XX	Exclude lines
F	First line
L	Last line
COLS	Columns on line
FS	Field split
LC	Line clear
G	Define group
U	Update group
DG	Delete group

The command field designates that you are back in the Prototyping Facility. Your changes are immediately reflected on the menu panel.

To simulate execution of the menu, enter Option **1** and **ID 123456**

An option of 1 transfers you to the Employee Add screen. Remember that level two prototyping converts the function to a **NEXT-PROGRAM-NAME-ID** to control this transfer.

Press Enter.

EMPLOYEE MENU SCREEN
1. ADD EMPLOYEE
2. LIST EMPLOYEES
3. DISPLAY EMPLOYEE
4. UPDATE EMPLOYEE
5. EXIT
OPTION 1
ID 123456

The Employee Add screen displays. Note that the ID entered on the Employee menu screen is now displayed on this screen. This is done by means of the active store, which is automatically created to keep this data.

Note the **%** symbols in the DATE OF EMPLOYMENT field. **%** is the undefined store character as shown on the Prototyping Facility menu under Presentation. The Prototyping Facility tried to map data to this field, but no data was defined for this field in the active store. Therefore, the undefined store character is displayed.

To define this data, enter the active presentation store to view its contents. To do this, enter the view command **VPS** on the command field, and press Enter]

```

EMPLOYEE ADD SCREEN

ID          123456
NAME
ADDRESS
PHONE
SEX
HOURLY RATE
HOURS PER WEEK
DATE OF EMPLOYMENT %%%%%%%%%
DATE OF BIRTH

.VPS

```

You now see the active store. It contains all the variables used as DBNAMEs on the menu and add panels you have so far prototyped.

Change the % characters in the EMPL-DOE field to an appropriate date value of **041890**. Then press PF3 to save these changes, and to return to the prototyping session.

```

EDIT --- *ACTIVE* STORE 001 OF 001 SIZE 000015 COL 01
COMMAND ==>          SCROLL ==> CSR

***** VALUE ***** DBNAME *****SUB LTH
***** ***** TOP OF DATA *****
000001          EMPL-CITY          001 025
000002          EMPL-DOB           001 006
000003 041890    EMPL-DOE          001 006
000004          EMPL-HOURLY-RATE    001 005
000005          EMPL-HOURS          001 004
000006 123456    EMPL-ID           001 006
000007          EMPL-NAME          001 025
000008          EMPL-PHONE         001 012
000009          EMPL-SEX           001 001
000010          EMPL-STATE         001 002
000011          EMPL-STREET        001 025
000012          EMPL-ZIP           001 005
000013 DISP     NEXT-PROGRAM-NAME-ID 001 005
000014 123456    XFER-EMPL-ID       001 006
000015 041890    XFER-TODAYS-DATE   001 006
***** ***** BOTTOM OF DATA *****

```

You return to the add panel from which you requested the active store. Initially, only the ID field is displayed. You can continue the simulation by entering any or all employee information on this screen.

When you created the screen definition (SD) for the Add panel, you specified that the Add screen always transfers to the Display screen. To simulate work flow, press Enter.

EMPLOYEE ADD SCREEN	
ID	123456
NAME	THOMAS DUNN_____
ADDRESS	1234 CORTLAND DR_____
	NAPERVILLE_____ IL 60565
PHONE	_____
SEX	-
HOURLY RATE	_____
HOURS PER WEEK	_____
DATE OF EMPLOYMENT	041890
DATE OF BIRTH	031460
*** ABOVE HIGHLIGHTED FIELDS IN ERROR ***	

CA Telon automatically transfers control to the Display screen by means of the NEXTPGM parameter in your SD, and CA Telon carries over all data entered on the Add screen.

041890 EMPLOYEE DISPLAY SCREEN		
EMPLOYEE ID	123456	
NAME	THOMAS DUNN	
ADDRESS	1234 CORTLAND DR NAPERVILLE IL 60565	
DATE OF BIRTH	031460	SEX
DATE OF EMPLOYMENT	041890	
HOURS PER WEEK	DEPARTMENT	%%
HOURLY RATE		
SALARY %%%%%%%%%%		
DISPLAY EMPLOYEE ID _____ UPDATE EMPLOYEE ID _____		

PFKEYS: 2-HOLD 3-END 4-ENDHOLD 6-MENU 7-PREV 8-NEXT		

You can continue your prototyping session by entering the appropriate data in the active store.

To end your prototyping session, type **/END** in the command area and press Enter.

Chapter 11: Creating the Program Definition

The final step in program development is to create the program definition. The program definition supplies processing information relevant to the entire program. The online program definition is known as the **screen definition** (SD) and the batch program definition is known as the **batch definition** (BD).

The screen definition and the batch definition perform basically the same functions for their programs. This chapter covers each of them separately to give full consideration to their specific requirements.

The **screen definition** defines program characteristics such as data access, custom code, and process control (which indicates where control passes when processing on the current screen is completed).

The **batch definition** defines program characteristics such as data access, custom code, work fields, and environment. In fact, the batch definition can be the whole program if no printed report is required. Otherwise, the batch definition builds on the report layout entered in the panel image and panel definition.

Create the Screen Definition

The procedure for creating a screen definition and data group depends on the system and data access method you are using. Procedures in this example are given for the following environments and data access types:

- CICS VSAM
- DB2 (CICS, TSO or IMS/DC)
- TSO/IMS/DC DL/I

To create a screen definition you must:

- Specify custom code editing
- Specify files to the program, also known as *adding or updating* a data group

After you add the data groups, you can preview the generated data to make sure that you have what you need. CA Telon lets you display the code on your screen as it will look before actually compiling the program.

To begin the screen definition, type **4** in the FUNCTION field of the TDF Main menu and press Enter.

```
TELON DESIGN FACILITY MAIN MENU *****
COMMAND ==> _____

FUNCTION: 4_

          1 -- USER PROFILE MAINTENANCE
          2 -- DATA ADMINISTRATION
          3 -- PANEL SPECIFICATION
          4 -- ONLINE PROGRAM DEFINITION
          5 -- BATCH PROGRAM DEFINITION
          6 -- PROTOTYPING FACILITY
          U -- UTILITIES
          X -- EXIT
```

CA Telon displays the Online Program Definition menu. Type **CR** in the FUNCTION field and **SD** in the ITEM field to create the screen definition for the add/update program.

The HEADER and ID are carried over from the Panel Definition menu, so you do not have to re-enter this information.

Press Enter.

```
ONLINE PROGRAM DEFINITION MENU *****
COMMAND ==> _____

FUNCTION: CR CR-CREATE UP-UPDATE PU-PURGE SH-SHOW LI-LIST
ITEM:     SD SD-SCREEN DR-DRIVER RD-REPORT ND-NONTERMINAL
          DG-DATA GROUP CC-CUSTOM CODE EN-ENVIRON

MEMBER NAME:
          HEADER TR
          ID UPDT_ TYPE _ (SD, DR, RD, ND)
          DESC _____

BASE DEFN : _____ (FOR CREATE - NAME OF BASE SD, DR, RD OR ND)

ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:
          1. ENVIRON CICS__ (CICS, IMS)
          2. CUSTCODE _____ (NAME OF CUSTOM CODE)
```

The Create/Update Program Definition screen displays. This screen allows you to designate the:

- Name of the program to follow this one
- Field where the program positions the cursor
- Programming language
- Screen size larger than 24 x 80

- Transfer work area copy member name(s)
- Other work area copy member name(s)
- Procedural COPY Code names for code added to the generated program
- Custom code members
- Extended screen attribute characteristics
- Additional specialized screen attributes

```

TRUPDT.SD CREATE SCREEN DEFINITION ** *****
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV IMS _ SCRN PARMS _
          STORED PROCEDURES _

GENERAL: DESC MVS CICS TARGET - COB VSAM ADD _____ REMARKS **DFLT**
*      NEXTPGM _____ CURSOR NAME _____ SIZE 24 X 80 LANG COB (COB/PLI)
*      CMPLOPT _____ IDENTIF _____ PROCEDR _____
DATA   XFERWKA _____
AREAS: _ WKAREA _____

OUTPUT:
A-100 _ OINIT1 _____ OINIT2 _____ CURSCUS _____
B-100 _ OUTTERM _____

INPUT:
P-100 _ PFKEYS _____
D-100 _ ININIT1 _____ ININIT2 _____

E-100 _ FLDEDIT _____
X-100 + SCREEN XFEDIT/SEGEDIT _ CONSIG _____
H-100 _ INTERM _____

MISC: _ SECTION _____
* PGM CUST _____

```

Specify Custom Code Editing

In the Add/Update program, you want the cursor to be placed on the first unprotected field (NAME), and then you want the program to unconditionally transfer to a display program. To do this, type **DISP** in the NEXTPGM field.

For this example, the program structure has been analyzed, and you have decided to place some custom code in three locations: OINIT2 (output initialization), ININIT1 (input initialization), and INTERM (input termination). Therefore, type **U** before each exit point to indicate that you are going to update the custom code point (type the code directly into CA Telon).

```
TRUPDT.SD CREATE SCREEN DEFINITION ** *****  
COMMAND ==>  
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV IMS _ SCRN PARMS _  
STORED PROCEDURES _  
  
GENERAL: DESC EMPLOYEE ADD/UPDATE SCREEN _ REMARKS _____  
*      NEXTPGM DISP CURSOR _____ SIZE 24 X 80 LANG COB (COB/PLI)  
*      CMPLOPT _____ IDENTIF _____ PROCEDR _____  
DATA   XFERWKA _____  
AREAS: _ WKAREA _____  
  
OUTPUT:  
A-100 _ OINIT1 _____ U OINIT2 _____ CURSCUS _____  
B-100 _ OUTTERM _____  
  
INPUT:  
P-100 PFKEYS ADD,ENT,1,2,3,4,5,6,OT _____  
D-100 U ININIT1 _____ ININIT2 _____  
  
E-100 _ FLDEDIT _____  
X-100 _ SCREEN XFEDIT/SEGEDIT _ CONSIS _____  
H-100 U INTERM _____  
  
MISC:  _ SECTION _____  
*      PGMJUST _____
```

The Custom Code Editor screen appears. You can put specialized code in a copybook and enter the name of the copybook at the custom code point, or you can type the code while in CA Telon itself, by using the custom code editor.

CA Telon's Custom Code editor allows you to update multiple members simultaneously. This feature allows copying code among members, viewing one while modifying another, and so on. The editor is very similar to the ISPF editors using similar line commands and command line features.

The following screen shows a screen allowing you to create three new custom code members for the program.

```

EDIT ---- TRUPDT.SD OINIT2 ***** SCREEN DEFINITION SAVED
COMMAND ==> SCROLL ==> CSR
=MBR=> TRUPDT.SD(OINIT2) 000000
.....
.....
.....
.....
.....
=MBR=> TRUPDT.SD(ININIT1) 000000
.....
.....
.....
.....
.....
=MBR=> TRUPDT.SD(INTERM) 000000
.....
.....
.....
.....
.....

```

At this point, enter the custom code to be inserted for output initialization (OINIT2) and input initialization (ININIT1). The CA Telon editor command, entered on any line(s) of a Custom Code Edit screen, creates a comment area enclosed by a rectangle of asterisks.

```

EDIT ---- TRUPDT.SD OINIT2 ***** SCREEN DEFINITION SAVED
COMMAND ==> SCROLL ==> CSR
=MBR=> TRUPDT.SD(OINIT2) 000000
*****
***** READ THE RECORD FROM THE FILE. IF IT EXISTS, MAP THE *
***** VALUES TO THE SCREEN (UPDATE MODE). IF IT DOESN'T MAP*
***** WITH ID AND CURRENT DATE ONLY (ADD MORE). *
*****
***** IF EMPLOYEE-STATUS-CODE = 'GE'
*****     MOVE XFER-EMPL-ID TO TPO-ID
*****     MOVE CURRENT-DATE TO TPO-DOE
*****     MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
*****     MOVE 'A' TO XFER-INDICATOR
***** ELSE
*****     MOVE 'U' TO XFER-INDICATOR.
*****
=MBR=> TRUPDT.SD(ININIT1) 000000
*****
***** IF WE ARE IN UPDATE MODE, RE-READ THE RECORD FROM THE*
***** FILE FOR UPDATING. *
*****
***** IF XFER-INDICATOR = 'U'
*****     PERFORM U-100-READ-TRGEMPL.
*****

```

Page down using PF8, as in ISPF, to see in the following screen. Type the custom code to be inserted for input terminations (INTERM). The input initialization still displays because it was paged down from the previous screen.

The custom code has now been entered in the three members.

OINIT2 code checks the status code after reading the database. If the segment was not found, the employee ID and current date are moved to the screen fields, and *Add Mode* (A in XFER-INDICATOR) is specified. Otherwise, *Update Mode* (U in XFER-INDICATOR) is specified. ININIT1 reads the database segment again to prepare for updating after the screen is returned to the program.

INTERM code then either updates or checks a record on the file.

```
=MBR=> TRUPDT.SD(ININIT1) 000000
*****
***** IF WE ARE IN UPDATE MODE, RE-READ THE RECORD FROM THE*
***** FILE FOR UPDATING. *
*****
***** IF XFER-INDICATOR = 'U'
***** PERFORM U-100-READ-TRGEMPL.

=MBR=> TRUPDT.SD(INTERM) 000000
*****
***** IF WE ARE IN UPDATE MODE, UPDATE THE RECORD ON THE *
***** FILE, OTHERWISE, INSERT THE RECORD ON THE FILE. *
*****
***** IF XFER-INDICATOR = 'U'
*****     PERFORM U-100-UPDATE-TRGEMPL.
***** ELSE
*****     PERFORM U-100-CREATE-TRGEMPL.
```

When you are done editing your custom code members, press PF3 to save them and return to the SD screen, shown below. You have returned from the custom code. Notice the default indication, ****DFLT****, beside each member that was created. CA Telon gave the members the default name. When you were adding these custom code members, you could have given the custom code members different names.

```

TRUPDT.SD CREATE SCREEN DEFINITION ** *****
COMMAND ==>
_ OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV IMS _ SCRN PARMS _
  STORED PROCEDURES _

GENERAL: DESC EMPLOYEE ADD/UPDATE SCREEN _ REMARKS *
*      NEXTPGM _____ CURSOR NAME _____ SIZE 24 X 80 LANG COB (COB/PLI)
*      CMPLOPT _____ IDENTIF _____ PROCEDR _____
DATA   XFERWKA _____
AREAS: WKAREA _____
OUTPUT:
A-100 _ OINIT1 _____ OINIT2 **DFLT** _ CURSCUS _____
B-100 _ OUTTERM _____

INPUT:
P-100  PFKEYS _____
D-100  ININIT1 **DFLT** _ ININIT2 _____

E-100  FLDEDIT _____
X-100  + SCREEN XFEDIT/SEGEDIT _ CONSIS _____
H-100  _ INTERM **DFLT**

MISC:  _ SECTION _____
*      PGMCUST _____

```

Create a Data Group

Now, the files need to be defined for the programs. CA Telon calls this *updating the data groups*. Type **U** in the DATA GROUP field and press Enter to create the data group. This is shown in the following screen.

```

TRUPDT.SD CREATE SCREEN DEFINITION ** *****
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV IMS _ SCRN PARMS _
          STORED PROCEDURES _

GENERAL: DESC MVS CICS TARGET - COB VSAM ADD _____ REMARKS **DFLT**
*      NEXTPGM _____ CURSOR NAME _____ SIZE 24 X 80 LANG COB (COB/PLI)
*      CMPLOPT _____ IDENTIF _____ PROCEDR _____
DATA   XFERWKA _____
AREAS: _ WKAREA _____

OUTPUT:
A-100 _ OINIT1 _____ OINIT2 **DFLT** CURSCUS _____
B-100 _ OUTTERM _____

INPUT:
P-100 _ PFKEYS _____
D-100 _ ININIT1 **DFLT** _ ININIT2 _____

E-100 _ FLDEDIT _____
X-100 _ SCREEN XFEDIT/SEGEDIT _ CONSID _____
H-100 _ INTERM **DFLT** _____

MISC: _ SECTION _____
* PGM CUST _____

```

CA Telon transfers you to the Update Data Group screen for this program (see below). Since we have not defined any files yet, you see a blank screen. If we had defined some files and the specific file access, you would see a list of what was defined.

This example uses DB2 Tables as the DBMS.

Note: The processing available for DB2 is essentially identical to other RDBMS supported by CA Telon, including CA IDMS Database SQL Option, CA Datacom Database SQL Option, SQL/400, and XDB.

Therefore, the command **DGADD TRGEMPL2.TAB** copies characteristics from an existing DB2 Table into this program. TRGEMPL2.TAB is the name of the DB2 Table defined to this CA Telon data group. The database administrator or a project leader at your installation defines the database.

```

UPDATE DATA GROUP ---- TRUPDT.SD ** USE "DGADD" COMMAND TO INIT DATA GROUP
COMMAND ==> DGADD TRGEMPL2.TAB                                SCROLL ==> CSR
          LABEL      REQUEST          KEY/WHERE          IGNORE      MORE
=====
***** ***** ***** BOTTOM OF DATA *****

```

Now, you see the screen after you have added the table definition to the program. TRGEMPL2 is the name of the row given to the DB2 that is being used.

The next step is to define the specific type of data access to use. That is, whether CA Telon is to read, update, write, or perform some other function on a file.

To specify the various data access calls needed for this program, type **I** in the left column to insert a line under the TRGEMPL2 Row. In the LABEL space, generic terms are used to indicate the functions to be performed. These terms will be the same no matter what DBMS you use.

```

UPDATE DATA GROUP ---- TRUPDT.SD ***** DATA GROUP SUCCESSFULLY EXPANDED
COMMAND ==>                                SCROLL ==> CSR
          LABEL      REQUEST      KEY/WHERE                                IGNORE      MORE
=====
TAB==> TRGEMPL2 TELON.TRGEMPL2 1
I    TRGEMPL2 @DUMMY @EMPL-ID
ROW==> XTRGEMPL @DUMMY @EMPL-ID TRGEMPL2
=====
***** ***** ***** BOTTOM OF DATA *****

```

Enter the four call routines as shown in the following routines:

- **CREATE:** Generates INSERT processing conditionally executed in INTERM custom code if the process is *add* mode.
- **UPDATE:** Generates UPDATE processing conditionally executed in INTERM custom code if the process is *update* Mode.
- **READ:** Generates a SELECT conditionally executed in ININIT1 custom code if the process is *update* mode.
- **OUTREAD:** Generates an automatic SELECT so the program can determine whether the row exists on the table (*add* or *update* modes). OUTREAD is executed before OINIT2 custom code.

The TDF allows the designer/programmer to preview the generated data access prior to generating the program. This saves time since the programmer need not wait for lengthy compiles to end before discovering that the call was not quite what was required.

```
UPDATE DATA GROUP ---- TRUPDT.SD MEMBER 001 OF 001 SIZE 000009 COL 01
COMMAND ==> SCROLL ==> CSR
      LABEL      REQUEST      KEY/WHERE      IGNORE      MORE
=====
TAB==> TRGEMPL2 TELON.TRGEMPL2_1
ROW=> TRGEMPL2 @DUMMY @EMPL-ID
      *****
      ***** CREATE
      ***** UPDATE
      ***** READ
      ***** AUTOEXEC OUTREAD
ROW=> XTRGEMPL @DUMMY @EMPL-ID TRGEMPL2
=====
***** ***** ***** BOTTOM OF DATA ***** *****
```

Preview Generated Data

To use the preview function, simply type **VIEW** beside the data access you want to preview and press Enter.

```
UPDATE DATA GROUP ---- TRUPDT.SD MEMBER 001 OF 001 SIZE 000009 COL 01
COMMAND ==>                                SCROLL ==> CSR
      LABEL      REQUEST      KEY/WHERE      IGNORE      MORE
=====
TAB==> TRGEMPL2 TELON.TRGEMPL2_1
ROW=> TRGEMPL2 @DUMMY @EMPL-ID
VIEW=> CREATE
      *****
      ***** UPDATE
      ***** READ
      ***** AUTOEXEC OUTREAD
ROW=> XTRGEMPL @DUMMY @EMPL-ID TRGEMPL2
=====
***** ***** ***** BOTTOM OF DATA ***** *****
```


The Preview screen appears. Notice that the preview of a CREATE data access generates an INSERT call, with the lengthy list of parameters. To view the rest of the parameters, press PF8.

```
VIEW ---- TRUPDT.SD *PREVIEW *****DATA GROUP SAVED
COMMAND ==>                                SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001 *      DEFAULT GENERATED DATA ACCESS
000002 *      DEFAULT CALL: ** U-100-CREATE-TRGEMPL2
000003      EXEC SQL INSERT
000004              INTO      TELON.TRGEMPL2
000005                      (EMPL-ID,
000006                      EMPL-NAME,
000007                      EMPL-DOB,
000008                      EMPL-SEX,
000009                      EMPL-PHONE,
000010                      EMPL-STREET,
000011                      EMPL-CITY,
000012                      EMPL-STATE,
000013                      EMPL-ZIP,
000014                      EMPL-DOE,
000015                      EMPL-DEPARTMENT,
000016                      EMPL-HOURLY-RATE,
000017                      EMPL-HOURS)
000018      VALUES (:DCLTRGEMPL2.EMPL-ID,
000019              :DCLTRGEMPL2.EMPL-NAME,
000020              :DCLTRGEMPL2.EMPL-DOB,
000021              :DCLTRGEMPL2.EMPL-SEX,
```

The following screen shows the rest of the parameters generated.

```
VIEW ---- TRUPDT.SD *PREVIEW MEMBER 001 OF 001 SIZE 000033 COL 07
COMMAND ==>                                SCROLL ==> CSR
000022              :DCLTRGEMPL2.EMPL-PHONE,
000023              :DCLTRGEMPL2.EMPL-STREET,
000024              :DCLTRGEMPL2.EMPL-CITY,
000025              :DCLTRGEMPL2.EMPL-STATE,
000026              :DCLTRGEMPL2.EMPL-ZIP,
000027              :DCLTRGEMPL2.EMPL-DOE,
000028              :DCLTRGEMPL2.EMPL-DEPARTMENT,
000029              :DCLTRGEMPL2.EMPL-HOURLY-RATE,
000030              :TRGEMPL2-EMPL-HOURLY-RATE-NN,
000031              :DCLTRGEMPL2.EMPL-HOURS,
000032              :TRGEMPL2-EMPL-HOURS-NN)
000033      END-EXEC.      :DCLTRGEMPL2.EMPL-PHONE,
***** ***** ***** BOTTOM OF DATA *****
```

At this point, the program is ready to be generated and compiled. The entire generated program would be about 2,000 lines of well-documented and well-structured COBOL code.

The design and construction process is virtually the same for all online programs, regardless of target environment (IMS, CICS), or DBMS (DL/I, DB2, VSAM).

Create the Batch Definition

The batch definition is the basic unit of a TDF-designed batch program. The batch definition defines all processing for your application. It includes all specifications necessary to export your CA Telon source code to the CA Telon generator for compilation into a completed COBOL or PL/I program.

If your program will produce a report, you must create a panel image (see Chapter 5) and a panel definition (see Chapter 6) before you create the batch definition. For programs that do not produce reports, the batch definition is enough to define the program.

The batch definition supplies information to CA Telon that is relevant to the entire program. This information includes:

- Basic characteristics
- Custom code
- Data access
- Extra work areas
- Report destination

Basic characteristics include a description, the programming language to be generated, the CA Telon release used, and the report size.

Custom code refers to any COBOL or PL/I code that you include in your program to perform logic beyond that which CA Telon normally performs.

Data access defines databases and data files that the programs access. It includes those automatic exec I/Os that are the same for all executions of the batch definition. It also includes any user exec I/O when your I/Os vary from one execution to the next based on specific conditions.

Extra work areas include the transfer work areas where calculations occur.

Report destination is the name of the file to which CA Telon writes the report.

The different parts of the batch definition can be entered in any order.

To begin the batch definition, type **5** in the FUNCTION field of the TDF Main menu and press Enter (shown following).

```

FUNCTION: 5_

      1 -- USER PROFILE MAINTENANCE
      2 -- DATA ADMINISTRATION
      3 -- PANEL SPECIFICATION
      4 -- ONLINE PROGRAM DEFINITION
      5 -- BATCH PROGRAM DEFINITION
      6 -- PROTOTYPING FACILITY
      U -- UTILITIES
      X -- EXIT

```

CA Telon displays the Batch Program Definition menu (see the following screen). The Batch Program Definition menu comes in two forms, short and long.

The USERMODE parameter under Update Session Controls determines the form of the menu. A USERMODE of 1 or spaces, the default, gives you the short form of the Batch Program Definition menu while a USERMODE of 2 gives you the long form.

```

BATCH PROGRAM DEFINITION MENU *****
COMMAND ==> _____

FUNCTION: __ CR-CREATE UP-UPDATE PU-PURGE SH-SHOW LI-LIST

ITEM:      __ BD-BATCH SP-STORED PROCEDURES PI-IMAGE PD-DEFIN
           DG-DATA GROUP CC-CUSTCODE EN-ENVIRON GP-GROUP

MEMBER NAME:
            HEADER _____
            ID _____ TYPE BD (BD, SP)
            DESC _____

BASE DEFN : _____ (FOR CREATE - NAME OF BASE BD OR SP)

ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:
      1. GROUP _____ (NAME OF GROUP)
      2. CUSTCODE _____ (NAME OF CUSTOM CODE)
      3. ENVIRON MVS _____ (MVS, STORED)

```

The long form requires you to fill in the SPECIFIC ITEM TO BE PROCESSED section in the following cases:

- If the FUNCTION is Update (UP) and the ITEM is Group (GP), you must enter the name of a logical group from the panel definition in **1. GROUP**.
- If the FUNCTION is **CR**, **UP**, or **PU** and the ITEM is custom code (**CC**), you must enter a value in **2. CUSTCODE**.

To access the short form, type **setmode** on the COMMAND line, as shown in the screen below. Press Enter.

The Setmode command allows you to switch back and forth between the two forms of the Batch Program Definition menu.

```
BATCH PROGRAM DEFINITION MENU *****
COMMAND ==> setmode_____
FUNCTION: __ CR-CREATE UP-UPDATE PU-PURGE SH-SHOW LI-LIST
ITEM:      __ BD-BATCH SP-STORED PROCEDURES PI-IMAGE PD-DEFIN
          DG-DATA GROUP CC-CUSTCODE EN-ENVIRON GP-GROUP

MEMBER NAME:
            HEADER _____
            ID _____ TYPE BD (BD, SP)
            DESC _____

BASE DEFN : _____ (FOR CREATE - NAME OF BASE BD OR SP)

ENTER VALUE FOR SPECIFIC ITEM TO BE PROCESSED:
1. GROUP _____ (NAME OF GROUP)
2. CUSTCODE _____ (NAME OF CUSTOM CODE)
3. ENVIRON MVS _____ (MVS, STORED)
```

Notice the BASE DEFN. field. Use this field to identify an existing batch definition to use as a base for the definition you are creating. Enter the HEADER and ID of the definition to be copied with **no** intervening spaces. You can use this parameter only when the FUNCTION is create. CA Telon then lets you modify the base to meet the needs of the current program.

Now, CA Telon displays the short form of the Batch Program Definition menu.

Notice that the only ITEM choice listed is now BD and that the SPECIFIC ITEM TO BE PROCESSED section is absent.

```
BATCH PROGRAM DEFINITION MENU *****
COMMAND ==> _____
FUNCTION: UP CR-CREATE UP-UPDATE PU-PURGE SH-SHOW LI-LIST
ITEM:      BD BD-BATCH SP-STORED PROCEDURES

MEMBER NAME:
            HEADER _____
            ID _____
            DESC _____

BASE DEFN : _____ (FOR CREATE - NAME OF BASE BD)
```

The FUNCTION, MEMBER NAME and DESCRIPTION fields are the same here as on the Panel Definition menu. If you move directly from the panel definition process to the batch definition, CA Telon carries over the HEADER and ID.

Type **CR** over UP in the FUNCTION field and leave **BD** in the ITEM field. On this screen, the batch definition (BD) is the only valid item.

Under MEMBER NAME, type the HEADER and the ID.

If you are creating a program that does not produce a report, you should also enter the description.

Press Enter.

```

BATCH PROGRAM DEFINITION MENU *****
COMMAND ==> _____

FUNCTION: CR CR-CREATE UP-UPDATE PU-PURGE SH-SHOW LI-LIST

ITEM:      BD BD-BATCH SP-STORED PROCEDURES

MEMBER NAME:
  HEADER TR____
  ID  BATC_
  DESC _____

BASE DEFN : _____ (FOR CREATE - NAME OF BASE BD)

```

CA Telon returns the Create Batch Definition screen (see the following screen). This screen allows you to designate all the information needed to complete the BD, including:

- Description
- CA Telon release version
- Report size
- Programming language
- Report destination
- COBOL copy members
- Work areas
- Custom code points
- Custom code to copy in
- Certain field lengths

```
TRBATC.BD CREATE BATCH DEFINITION *****
COMMAND ==>-----
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS _
          STORED PROCEDURES _

GENERAL: DESC -----
*        LANGVL 4.1 SIZE 60 X 133 LANG COB (COB/PLI)
*        STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*        USER SORTS
*        EMPLOPT ----- IDENTIF ----- PROCEDR -----
FILES:   RPTDEST -----
*        COBFCPY:SELECT ----- FILEDEF -----
AREAS:   WKAREA -----

Q-1000 _ INIT1 ----- INIT2 -----
C-1000 _ GETTRAN -----

A-1000 _ PRCTRAN -----

T-1000 _ TERM -----

MISC:   SECTION -----
*        PGM CUST -----
LINKAGE: PARMS -----
```

Review the Panel Definition

In addition to providing access to all batch definition fields that must be entered, the Create Batch Definition screen lets you review and, if desired, change the panel definition.

To access the panel definition, type **U** in the PANEL DEF field and press Enter.

```

TRBATC.BD CREATE BATCH DEFINITION *****
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF U ENV MVS _
          STORED PROCEDURES _

GENERAL: DESC EMPLOYEE REPORT _ REMARKS _
*        LANGLVL 4.1 _ SIZE 60 X 133 LANG COB (COB/PLI) _
*        STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*        USER SORTS
*        CMPTOPT _ IDENTIF _ PROCEDR _
FILES:   RPTDEST REPORT1 _
*        COBCPY: SELECT _ FILEDEF _
AREAS:   WKAREA _

Q-1000 _ INIT1 _ INIT2 _
C-1000 _ GETTRAN _

A-1000 _ PRCTRAN _

T-1000 _ TERM _

MISC:   SECTION _
*        PGM CUST _
LINKAGE: PARMS _

```

With the information specific to your application, the Update Batch Panel Fields screen appears.

To modify this information, simply type over the information, such as the position and length of the fields.

Note: In batch applications, the LN/COL/LTH information indicates the position of the field or literal within its logical group, as specified in the panel definition.

A + in the **MORE** column indicates that CA Telon has additional information about the field or group. To access this information, type **U** in the update column and press Enter. This allows you to view the menus one level below. Other valid entries in the U column are **I** to insert a blank line below and **D** to delete the line.

After you make all necessary changes and verify the accuracy of the information, press PF3 to return to the Batch Definition screen.

```

TRBATC.PD UPDATE PANEL FIELDS *****
COMMAND ==> PAGE 01
LINE 001 COL 002 ***** SIZE 053 133
LINE ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+
0001 *** GROUP NAME=TITLE TYPE=TOPPAGE *****
0002 EMPLOYEE REPORT
0003
GP TYPE OR
U LN COL LTH USE **NAME** *FLDTYPE* ***** DBNAME OR TEXT ***** MORE
      GP  TITLE  TOPPAGE
      GP  EMPINFO  DETAIL
01 002 006 OU  ID  EMPL-ID
01 018 025 OU  NAME  EMPL-NAME
01 054 005 OU  HRWEEK  EMPL-HOURS
      +
02 018 025 OU  STREET  EMPL-STREET
03 018 025 OU  CITY  EMPL-CITY
      GP  ENDINFO  SUMMARY
01 046 003 OU  TOTEMP
      +

```

Upon return to the Update Batch Definition screen, CA Telon displays the message PANEL DEFINITION SAVED in the upper-right corner.

```

TRBATC.BD UPDATE BATCH DEFINITION *****PANEL DEFINITION SAVED
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS _
          STORED PROCEDURES _

GENERAL: DESC TEST PROBLEM 891 TELON _ REMARKS _
*      LONGLVL 4.1 _ SIZE 60 X 133 LANG COB (COB/PLI)
*      STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*      USER SORTS
*      EMPLOPT _ _ IDENTIF _ _ PROCEDR _
FILES:  RPTDEST _
*      COBFCPY:SELECT _ FILEDEF _
AREAS:  WKAREA _

Q-1000 _ INIT1 _ _ INIT2 _
C-1000 _ GETTRAN _

A-1000 _ PRCTRAN _

T-1000 _ TERM _

MISC:  _ SECTION _
*      PGM CUST _
LINKAGE: PARMS _

```


Create a Data Group

You have established your batch definition and now need to complete the TDF process by creating your data group and entering any custom code and environment information. Begin this process on the Update Batch Definition screen (see the following screen).

You must first specify the report destination. Type the name of the data set that is to hold the report in the RPTDEST field. The default is REPORT. If you do not specify a report destination, it is difficult to distinguish your own output.

Then access the data group screen by typing **U** in the DATA GROUP field. Press Enter.

```

TRBATC.BD CREATE BATCH DEFINITION *****BATCH ENV SAVED
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS _
          STORED PROCEDURES _

GENERAL: DESC TEST PROBLEM 891 TELON _ REMARKS _
*       LANGLVL 4.1 _ SIZE 60 X 133 LANG COB (COB/PLI) _
*       STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*       USER SORTS
*       CMPLOPT _ IDENTIF _ PROCEDR _
FILES:  RPTDEST REPORT1 _
*       COBFCPY:SELECT _ FILEDEF _
AREAS:  _ WKAREA _

Q-1000 U INIT1 _ INIT2 _
C-1000 _ GETTRAN _

A-1000 _ PRCTRAN _

T-1000 _ TERM _

MISC:  _ SECTION _
*      _ PGM CUST _
LINKAGE: PARMS _

```

CA Telon will return with the Update Data Group screen. It is the first screen in the process of adding a data group to your application.

Since we have not defined any files yet, you see a blank screen. If we had defined some files and the specific file access, you would see a list of what was defined.

```

UPDATE DATA GROUP ---- TRBATC.BD ***** BATCH DEFINITION SAVED
COMMAND ==>                                SCROLL ==> CSR
          LABEL      REQUEST      KEY/WHERE      IGNORE
=====
***** ***** ***** BOTTOM OF DATA *****

```

Type **DGADD** and the name of the DB2 table you want to add on the COMMAND line. Press Enter.

The DGADD command copies characteristics of the DB2 table into this program. In effect, the command adds the Employee table to your data group. Of course, the name of the data group varies by database and by installation. Consult your installation's CA Telon coordinator if you are unsure of a table or data set name.

```

UPDATE DATA GROUP ---- TRBATC.BD ***** BATCH DEFINITION SAVED
COMMAND ==>DGADD TRGEMPL1.TAB                SCROLL ==> CSR
      LABEL      REQUEST      KEY/WHERE                IGNORE
=====
***** ***** ***** BOTTOM OF DATA *****

```

The screen now shows two lines representing the SQL table that can be accessed by the batch definition (see the following screen). The far-left column identifies the SQL tables and their corresponding rows. The LABEL column shows the name of the table and row. The REQUEST column specifies the type of I/O associated with the row.

Valid entries in the REQUEST column are:

Parameter	Description
TRANSACT	Generates a COPY statement for the I/O area of the segment.
DEFINE	Defines the REC/SEG/ROW to the TDF. CA Telon will process it in the generation procedure.
DUMMY	Indicates that REC/SEG/ROW is not being used.
@DEFINE	Indicates that REC/SEG/ROW is being used, but CA Telon is controlling it.
@DUMMY	Indicates the default. CA Telon thinks the REC/SEG/ROW is being used. This flips to @DEFINE when you insert the type of data access you want to use.

```

UPDATE DATA GROUP ---- TRBATC.BD SIZE 000007 COL 01
COMMAND ==>                SCROLL ==> CSR
      LABEL      REQUEST      KEY/WHERE                IGNORE
=====
TAB=> TRGEMPL1 TELON.TRGEMPL1
ROW=> TRGEMPL1 @DUMMY +
***** ***** ***** BOTTOM OF DATA *****

```

Type **I** (insert) in the first position of the ROW statement for TRGEMPL1 and press Enter.

```

UPDATE DATA GROUP ---- TRBATC.BD SIZE 000007 COL 01
COMMAND ==>                SCROLL ==> CSR
      LABEL    REQUEST      KEY/WHERE                IGNORE
=====
TAB==> TRGEMPL1 TELON.TRGEMPL1
I TRGEMPL1 @DUMMY +
***** ***** ***** BOTTOM OF DATA *****

```

CA Telon adds a line to the Update Data Group screen.

Enter **TRANSACT** in the REQUEST column. CA Telon will generate an SQL statement for this table.

Type **U** in the left column to update this I/O statement and press Enter.

```

UPDATE DATA GROUP ---- TRBATC.BD SIZE 000007 COL 01
COMMAND ==>                SCROLL ==> CSR
      LABEL    REQUEST      KEY/WHERE                IGNORE
=====
TAB==> TRGEMPL1 TELON.TRGEMPL1
ROW=> TRGEMPL1 @DUMMY @? +
U ' ' ' ' TRANSACT
***** ***** ***** BOTTOM OF DATA *****

```

The updated Update Data Group screen appears (see the following screen). Enter **U** next to the REC field to update the record.

```

UPDATE DATA GROUP ---- TRBATC.BD SIZE 000007 COL 01
COMMAND ==>                SCROLL ==> CSR
      LABEL    REQUEST      KEY/WHERE                IGNORE
=====
TAB==> TRGEMPL1
U ROW=> TRGEMPL1 @DEFINE @EMPL-ID +
0001 AUTOEXEC TRANSACT
***** ***** ***** BOTTOM OF DATA *****

```

CA Telon issues a read request, using the inherited WHERE clause (@). To update the request, enter U in the sequence field.

```

UPDATE DATA GROUP ---- TRBATC.BD SIZE 000007 COL 01
COMMAND ==>                SCROLL ==> CSR
=====
      LABEL      REQUEST      KEY/WHERE                IGNORE
=====
TAB==> TRGEMPL1 TELON.TRGEMPL1
ROW=>  TRGEMPL1 @DEFINE @EMPL-ID
U 0001 READ      @
ROW=>  EXTRA1  @DEFINE @EMPL-NAME      TRGEMPL1
0001  READ      @
ROW=>  UPDATE  @DUMMY @EMPL-ID                TRGEMPL1
***** ***** ***** BOTTOM OF DATA *****

```

The Update Request screen appears (see the following screen.)

The KEY/WHERE, KEYCOLS, and OPCODE fields are set by the @, inherited from Option 2. These three automatic settings determine the WHERE clause that is constructed for each column (for example, WHERE [KEYCOLS OPCODE :KEY/WHERE]).

Enter any character in the PREVIEW field to see the SQL code that will be generated in this program.

```

TRBATC.BD ** U-100-READ-TRGEMPL1 *** *****
COMMAND ==>
OPTIONS ==> PREVIEW X
GENERAL: KEY OR @EMPL-ID
*      WHERE
*
*
*      IGNORE
*      IOAREA @
DB2/SQL: SENCOLS @
*
*
*      ORDERBY
*
*      GROUPBY
*      HAVING
*
*
*      KEYCOLS @EMPL_ID
*
*      OPCODE @
CUSTOM CODE: _ CPYCALL _ CPYINIT
*           _ CPYKEY _ CPYTERM

```

The Preview screen appears. The 13 columns in the appropriate TLNROW have been selected. The three-part WHERE clause has been constructed from the KEY/WHERE, KEYCOLS, and OPCODE parameters. The columns selected automatically have the appropriate host variable as the destination of the actual data in the program. Note that a nullable column also requires an indicator variable as well as a host variable and is therefore generated by CA Telon.

```

VIEW ---- TRBATC.BD *PREVIEW ***** DATA GROUP SAVED
COMMAND ==> SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001 *      DEFAULT GENERATED DATA ACCESS
000002 *      DEFAULT CALL: ** U-100-READ-TRGEMPL1 ***
000003      EXEC SQL
000004              SELECT  EMPL_ID,
000005                      EMPL_NAME,
000006                      EMPL_DOB,
000007                      EMPL_SEX,
000008                      EMPL_PHONE,
000009                      EMPL_STREET,
000010                      EMPL_CITY,
000011                      EMPL_STATE,
000012                      EMPL_ZIP,
000013                      EMPL_DOE,
000014                      EMPL_DEPARTMENT,
000015                      EMPL_HOURLY_RATE
000016                      EMPL_HOURS
000017              INTO
000018                      :DCLTRGEMPL1.EMPL-ID,
000019                      :DCLTRGEMPL1.EMPL-NAME,
000020                      :DCLTRGEMPL1.EMPL-DOB,
000021                      :DCLTRGEMPL1.EMPL-SEX,
000022                      :DCLTRGEMPL1.EMPL-PHONE,
000023                      :DCLTRGEMPL1.EMPL-STREET,
000024                      :DCLTRGEMPL1.EMPL-CITY,
000025                      :DCLTRGEMPL1.EMPL-STATE,
000026                      :DCLTRGEMPL1.EMPL-ZIP,
000027                      :DCLTRGEMPL1.EMPL-DOE,
000028                      :DCLTRGEMPL1.EMPL-DEPARTMENT,
000029                      :DCLTRGEMPL1.EMPL-HOURLY-RATE
000030                      :TRGEMPL1-EMPL-HOURLY-RATE-NN,
000031                      :DCLTRGEMPL1.EMPL-HOURS
000032                      :TRGEMPL1-EMPL-HOURS-NN
000033              FROM      TELON.TRGEMPL1
000034              WHERE      (EMPL_ID = :DCLTRGEMPL1.EMPL-ID)
000035              END-EXEC
***** ***** BOTTOM OF DATA *****

```

Press PF3 to return to the Update Request screen (see the screen below). To restrict the number of columns selected by this call, use the SENCOLS parameter. An @ indicates to *inherit* all columns in the TLNROW. To obtain a list of all columns available, enter a **U** to the far right of the SENCOLS parameter.

```
TRBATC.BD ** U-100-READ-TRGEMPL1 *** ***** END PROCESSING PERFORMED
COMMAND ==> _____
OPTIONS ==> PREVIEW _____
GENERAL: KEY OR @EMPL_ID _____
*          WHERE _____
*          _____
*          _____
*          IGNORE _____
*          IOAREA @ _____
DB2/SQL: SENCOLS @ _____ u
*          _____
*          _____
*          _____
*          ORDERBY _____ -
*          _____
*          GROUPBY _____ -
*          HAVING _____
*          _____
*          _____
*          KEYCOLS @EMPL_ID _____
*          _____
*          OPCODE @ _____
CUSTOM CODE: _ CPYCALL _____ CPYINIT _____
*          _ CPYKEY _____ _ CPYTERM _____
```

The Select Columns screen appears (see the following screen). Enter an **S** to the left of all columns to be selected for this call.

```

SELECT COLUMNS *****
COMMAND ==> _____ PAGE 01
TABLE NAME TELON.TRGEMPL1 ROW NAME TRGEMPL1

S COLUMN NAME ALIAS ***** KY/AC TYPE LTH/DEC -NULL *****
* ***** **/** ***** *
s EMPL_ID @EMPL-ID Y CHAR 6
Y
s EMPL_NAME @EMPL-NAME CHAR 25 Y
s EMPL_DOB @EMPL-DOB DEC 6
Y
s EMPL_SEX @EMPL-SEX CHAR 1
Y
s EMPL_PHONE @EMPL-PHONE CHAR 10 Y
- EMPL_STREET @EMPL-STREET CHAR 25 Y
- EMPL_CITY @EMPL-CITY CHAR 25 Y
- EMPL_STATE @EMPL-STATE CHAR 2 Y
- EMPL_ZIP @EMPL-ZIP CHAR 5
Y
- EMPL_DOE @EMPL-DOE DEC 6
Y
- EMPL_DEPARTMENT @EMPL-DEPARTMENT CHAR 3
Y
s EMPL_HOURLY_RATE @EMPL-HOURLY-RATE DEC 5 2
N
- EMPL_HOURS @EMPL-HOURS DEC 4 1 N

```

Press PF3 to return to the Update Request screen.

The SENCOLS parameter now specifies all the required columns. (To reset the SENCOLS parameter to all columns, just enter an @ in the first character.)

```

TRBATC.BD ** U-100-READ-TRGEMPL1 *** **** 06 SELECTED COLUMNS APPENDED TO LIST
COMMAND ==>
OPTIONS ==> PREVIEW x
GENERAL: KEY OR @EMPL-ID
*
*   WHERE
*
*
*   IGNORE
*   IOAREA @
DB2/SQL: SENCOLS
EMPL_ID,EMPL_NAME,EMPL_DOB,EMPL_SEX,EMPL_PHONE,_EMPL_HOURLY_RATE_
*
*
*   ORDERBY
*
*   GROUPBY
*   HAVING
*
*
*   KEYCOLS @EMPL_ID
*
*   OPCODE @
CUSTOM CODE: _ CPYCALL _ _ CPYINIT
*             _ CPYKEY _ _ CPYTERM

```

Once again, enter any character in the PREVIEW field. Invoke the Preview function to check the SQL code (see the following screen).

Only the columns selected are accessed and moved to the appropriate host variable, as required, for this call only.

```

VIEW ---- TRBATC.BD *PREVIEW ***** DATA GROUP SAVED
COMMAND ==>
***** ***** TOP OF DATA *****
000001 *   DEFAULT GENERATED DATA ACCESS
000002 *   DEFAULT CALL: ** U-100-READ-TRGEMPL1 ***
000003   EXEC SQL
000004           SELECT  EMPL_ID,
000005                      EMPL_NAME,
000006                      EMPL_DOB,
000007                      EMPL_SEX,
000008                      EMPL_PHONE,
000009                      EMPL_HOURLY_RATE
000010           INTO
000011                      :DCLTRGEMPL1.EMPL-ID,
000012                      :DCLTRGEMPL1.EMPL-NAME,
000013                      :DCLTRGEMPL1.EMPL-DOB,
000014                      :DCLTRGEMPL1.EMPL-SEX,
000015                      :DCLTRGEMPL1.EMPL-PHONE,
000016                      :DCLTRGEMPL1.EMPL-HOURLY-RATE
000017           FROM      TELON.TRGEMPL1
000018           WHERE      (EMPL_ID = :DCLTRGEMPL1.EMPL-ID)
000021           END-EXEC.

```


Press PF3 to return to the Update Group screen (see the following screen).

UPDATE DATA GROUP ---- TRBATC.BD SIZE 000007 COL 01			
COMMAND ==>		SCROLL ==> CSR	
	LABEL	REQUEST	KEY/WHERE
			IGNORE
TAB=>	TRGEMPL1		
ROW=>	TRGEMPL1	@DEFINE @EMPL-ID	
	+		
0001	AUTOEXEC	TRANSACTION	
SEQ=>	REPORT1		
REC=>	REPORT1	@DUMMY	
***** ***** ***** BOTTOM OF DATA ***** *****			

As the last step in this process, enter **DGADD REPORT1.SEQ** on the COMMAND line. Press Enter. This defines your report file in the data group.

Enter Custom Code

Custom code is used to add logic beyond that which CA Telon usually performs. Batch programs provide four distinct areas to add custom code: **Q-1000**, **C-1000**, **A-1000**, and **T-1000**. They appear in the CUSTOM section in the lower right of the Update Batch Definition screen. Each custom code area listed represents an exit point in CA Telon. Refer to the processing flowcharts for the exact location of the points.

This example shows how to update the Q-1000 INIT1 custom code module, which is executed after the program opens the files. INIT1, the initial call on the data, often requires custom code, whereas subsequent calls are automatic. In any case, follow the same steps to create other custom code members.

Type **U** to the right of Q-1000 and press Enter.

```

TRBATC.BD UPDATE BATCH DEFINITION *** *****
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS _
STORED PROCEDURES _
GENERAL: DESC TEST PROBLEM 891 TELON _ REMARKS _
*   LANGLVL 4.1 _ SIZE 60 X 133 LANG COB (COB/PLI)
*   STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*   USER SORTS
*   CMPLOPT _ IDENTIF _ PROCEDR _
FILES:  RPTDEST REPORT1 _
*   COBCPY:SELECT _ FILEDEF _
AREAS:  WKAREA _

Q-1000  _ INIT1 _ INIT2 _
C-1000  _ GETTRAN _

A-1000  _ PRCTRAN _

T-1000  _ TERM _

MISC:  _ SECTION _
*   PGM CUST _
LINKAGE: PARMS _

```

Note: You cannot use the CUSTOM CODE field on the FUNCTION line until you have created some of the custom code modules.

The INIT1 Custom Code Edit screen appears. You will enter all the required custom code and informative comments about it on this screen.

```

EDIT ---- TRBATC.BD INIT1      MEMBER 001 OF 001      SIZE 000000 COL 07
COMMAND ==>
***** ***** TOP OF DATA *****
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
***** ***** BOTTOM OF DATA *****

```

[illegible]

```

EDIT ---- TREATC.BD INIT1          MEMBER 001 OF 001          SIZE 000006 COL 07
COMMAND ==>                                SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001
000002 *****
000003 *
      *
000004 *
      *
000005 *
      *
000006 *****
***** ***** BOTTOM OF DATA *****

```

- Factors to consider later
- Secret developer information

An example is shown in the following screen.

```

EDIT ---- TRBATC.BD INIT1      MEMBER 001 OF 001      SIZE 000006 COL 07
COMMAND ==>                      SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001
000002 *****
000003 *
      *
000004 * THIS COMMENT BOX EXPLAINS THE CUSTOM CODE THAT WILL FOLLOW *
000005 *
      *
000006 *****
***** ***** BOTTOM OF DATA *****

```

The CA Telon Editor responds to normal ISPF commands. For example, to list five more lines on the screen, type **I5** (insert 5) on the line below which you want the lines inserted. Press Enter. You can insert any number of lines you want.

```

EDIT ---- TRBATC.BD INIT1      MEMBER 001 OF 001      SIZE 000006 COL 07
COMMAND ==>                      SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001
000002 *****
000003 *
      *
000004 * THIS COMMENT BOX EXPLAINS THE CUSTOM CODE THAT WILL FOLLOW *
000005 *
      *
I50006 *****
***** ***** BOTTOM OF DATA *****

```

After you have entered your comments, type the COBOL or PL/I code directly into CA Telon and edit it on this screen until you are satisfied with it (see the next screen).

When you have finished coding on this screen, press PF3 to return to the Update Batch Definition screen.

```

EDIT ---- TRBATC.BD INIT1      MEMBER 001 OF 001      SIZE 000006 COL 07
COMMAND ==>                                SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001
000002 *****
000003 *
      *
000004 * THE COMMENT BOX EXPLAINS THE CUSTOM CODE THAT WILL FOLLOW *
000005 *
      *
000006 *****
      MOVE ZEROS TO MY-COUNTER
      .
      .
      .
      .
***** ***** BOTTOM OF DATA *****

```

The Update Batch Definition screen displays the message **END PROCESSING PERFORMED** in the upper right.

If you did not name the Custom Code module, ****DFLT**** appears to the right of **INIT1**.

```

TRBATC.BD UPDATE BATCH DEFINITION *** *****END PROCESSING PERFORMED
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS _
STORED PROCEDURES
GENERAL: DESC TEST PROBLEM 891 TELON _ REMARKS _
*   LONGLVL 4.1 SIZE 60 X 133 LANG COB (COB/PLI)
*   STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*   USER SORTS
*   CMPOPT _ IDENTIF _ PROCEDR _
FILES: RPTDEST REPORT1 _
*   COBCPY: SELECT _ FILEDEF _
AREAS: _ WKAREA _

Q-1000 _ INIT1 **DFLT** _ INIT2 _
C-1000 _ GETTRAN _

A-1000 _ PRCTRAN _

T-1000 _ TERM _

MISC: _ SECTION _
*   PGM CUST _
LINKAGE: PARMS _

```

Specify Environment

You must now specify environment characteristics for your application. Type **U** in the ENV field on the Update Batch Definition screen.

```

TRBATC.BD UPDATE BATCH DEFINITION *****END PROCESSING PERFORMED
COMMAND ==> _____
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS U
STORED PROCEDURES
GENERAL: DESC TEST PROBLEM 891 TELON _____ REMARKS _____
*      LANGLVL 4.1 SIZE 60 X 133 LANG COB (COB/PLI)
*      STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*                      USER SORTS
*      CMPLOPT _____ IDENTIF _____ PROCEDR _____
FILES:  RPTDEST _____
*      COBFCPY:SELECT _____ FILEDEF _____
AREAS:  WKAREA _____

Q-1000 _ INIT1 **DFLT** _ INIT2 _____
C-1000 _ GETTRAN _____

A-1000 _ PRCTRAN _____

T-1000 _ TERM _____

MISC:  SECTION _____
*      PGM CUST _____
LINKAGE: PARMS _____

```

On the Update Batch Environment screen, enter **DB2** in the DBMS parameter to indicate the database and **Y** in the TRACE parameter to specify that you want trace logic generated for the purpose of debugging during testing.

Press PF3 to return to the Update Batch Definition screen.

```

TRBATC.BD UPDATE BATCH ENV *****
COMMAND ==> _____

GENERAL: TRACE          Y (Y/N)
*      DBMS            DB2
*      PGMNAME         _____

LINKAGE: LNKCOPY _____
*      USGCOPY         _____

DLI: GENPCBS _____ (Y/N)
*      DLIWGT          (Y/N)
*      PSBNAME         _____

```

The Update Batch Definition screen returns displaying the message BATCH ENV SAVED, in the upper-right corner. This indicates that the batch environment information was added.

```

TRBATC.BD UPDATE BATCH DEFINITION *****BATCH ENV SAVED
COMMAND ==>
OPTIONS ==> CUSTOM CODE _ DATA GROUP _ PANEL DEF _ ENV MVS _
STORED PROCEDURES _
GENERAL: DESC TEST PROBLEM 891 TELON _ REMARKS _
*          LANGLVL 4.1 _ SIZE 60 X 133 LANG COB (COB/PLI)
*          STRUCTURE: + STANDARD _ MAIN SORT _ MERGE _ MATCH
*          _ USER SORTS
*          CMPLOPT _ IDENTIF _ PROCEDR _
FILES: RPTDEST _
*      COBCPY: SELECT _ FILEDEF _
AREAS: _ WKAREA _

Q-1000 _ INIT1 **DFLT** _ INIT2 _
C-1000 _ GETTRAN _

A-1000 _ PRCTRAN _
T-1000 _ TERM _

MISC: _ SECTION _
* PGM CUST _
LINKAGE: PARMS _

```

At this point, you can go back to any of the previous steps in creating the batch definition and check or alter your work.

When you are sure that everything has been entered correctly, you can generate, compile, and link the program and test it.

Chapter 12: Initializing Applications

This chapter shows you how to initialize your applications. First, it shows you how to initialize your applications in an IMS/DC environment. Defining COPY members, creating a TSO test PCB, and allocating test databases are discussed in detail.

Next, it shows you how to initialize your applications in a CICS environment.

Finally, this chapter discusses the transfer work area, an area required for passing data between programs in your application and/or from one iteration to another of the same program.

Note: Information on initializing application for other target environments may be found in the corresponding target option documentation.

IMS/DC

You must perform four initialization tasks to compile and test a new CA Telon application in an IMS/DC environment:

1. Define CA Telon application COPY members
2. Define database segment COPY members
3. Create a TSO test PSB
4. Allocate test databases

The first two tasks are necessary for compiling the generated programs; the second two tasks are required for testing the programs using TSO. (None of these tasks are prerequisite to designing screens using the CA Telon Design Facility.)

A discussion of these four tasks follows.

Task 1: Define CA Telon Application COPY Members

To compile a generated program, these five COPY members (INCLUDE members for PL/I) are required:

- **hhWKAREA** – Defines the work variables used in the application (where hh is the HEADER used to identify the application).
- **hhUPDTA** – Defines the work area for generated DL/I update processing (hh is the application HEADER).

- **hhPCBs** – Applies to programs being tested under TSO. Defines the PCB descriptions to be included in the programs (hh is the application HEADER).
- **hhPROC** – Applies to programs being tested under TSO. Defines the PSB structures to be included in the programs (hh is the application HEADER).
- **Transfer work area COPY member** – Defines a program work area in which data is transferred from one program to another or from iteration to iteration of the same program. Its name is defined using the XFERWKA parameter on the Update Program Definition Defaults screen, the Create/Update Screen Definition screen, or the Create/Update IMS/DC Report Definition screen. The transfer Work Area is discussed in detail under the subject "Transfer work area Definition" later in this chapter.

Note: The names of the first four COPY members are the CA Telon defaults; they can be defined differently for your installation, as described in the *Design Facility Reference*.

hhWKAREA

All programs in an application use a common application work area defining the common variables used in that application. This work area includes some standard CA Telon fields, as well as any application-specific fields used during processing. Although you can specify a program work area as well as this application work area, combining all fields that are common in an application into this area improves the efficiency and consistency of your code.

The application work area is not identified explicitly in the non-procedural statements, but is included automatically for each program generated. You must name its copy library hhWKAREA, where hh is the application HEADER.

The work area can take any form you want. It must include all system reserved fields (described in Chapter 14) as application work area variables, and can include any number of other (for example, application specific) fields. The area's definition must be a 01-level item (having any user-defined name). The definition of the required application work area fields is shown below, first for COBOL and then for PL/I.

For COBOL:

```

01 WORKAREA.
  05 ERROR-MESSAGE-NOHIT          PIC X(28)
    VALUE '*** NO OPTION SELECTED ***'.
  05 ERROR-MESSAGE-MULTHIT        PIC X(37)
    VALUE '*** MORE THAN ONE OPTION SELECTED ***'.
  05 ERROR-MESSAGE-HIGHLIGHT      PIC X(41)
    VALUE '*** ABOVE HIGHLIGHTED FIELDS IN ERROR ***'.
  05 ERROR-MESSAGE-SELECT-LINE-NO PIC X(28)
    VALUE '*** INVALID LINE NUMBER ***'.
  05 ERROR-MESSAGE-HELP          PIC X(33)
    VALUE '*** RETURN FROM HELP FUNCTION ***'.
  05 ERROR-MESSAGE-HOLD          PIC X(33)
    VALUE '*** RETURN FROM HOLD FUNCTION ***'.
  05 ERROR-MESSAGE-RESUME        PIC X(27)
    VALUE '*** NOT IN HOLD SESSION ***' .
  05 ERROR-MESSAGE-HOLD-ISRT      PIC X(33)
    VALUE '*** ALREADY IN HOLD SESSION ***' .
  05 MORE-LITERAL                PIC X(15)      VALUE '***MORE PAGES *'.
  05 NO-MORE-LITERAL             PIC X(15)      VALUE '* END OF LIST *'.
  05 ERROR-REQ-CHAR              PIC X          VALUE '*'.
  05 PRINT-ERROR-FLAG            PIC X          VALUE SPACES.
  05 HELP-CHAR                   PIC X          VALUE '?'.
  05 HELP-FIELD-PGM-ID           PIC X(4)       VALUE 'CTDH'.
  05 HELP-SCREEN-PGM-ID          PIC X(4)       VALUE 'CTDH'.
  05 HELP-TABLE-LIMIT            PIC 9(2)       COMP VALUE 10.
  05 NO-MAP-FLAG                 PIC X          VALUE 'N'.

```

For PL/I:

```
DCL 1 WORKAREA,
  05 ERROR_MESSAGE_NOHIT          CHAR(28)
    INIT('*** NO OPTION SELECTED ***'),
  05 ERROR_MESSAGE_MULTHIT        CHAR(37)
    INIT('*** MORE THAN ONE OPTION SELECTED ***'),
  05 ERROR_MESSAGE_HIGHLIGHT      CHAR(50)
    INIT('*** ABOVE HIGHLIGHTED FIELDS IN ERROR ***'),
  05 ERROR_MESSAGE_SELECT_LINE_NO CHAR(28)
    INIT('*** INVALID LINE NUMBER ***'),
  05 ERROR_MESSAGE_HELP           CHAR(33)
    INIT('*** RETURN FROM HELP FUNCTION ***'),
  05 ERROR_MESSAGE_HOLD           CHAR(33)
    INIT('*** RETURN FROM HOLD FUNCTION ***'),
  05 ERROR_MESSAGE_RESUME         CHAR(27)
    INIT('*** NOT IN HOLD SESSION ***'),
  05 ERROR_MESSAGE_HOLD_ISRT      CHAR(31)
    INIT('*** ALREADY IN HOLD SESSION ***'),
  05 MORE_LITERAL                 CHAR(15) INIT('* MORE PAGES *'),
  05 NO_MORE_LITERAL              CHAR(15) INIT('* END OF LIST *'),
  05 ERROR_REQ_CHAR               CHAR(1) INIT('*'),
  05 PRINT_ERROR_FLAG             CHAR(1) INIT(' '),
  05 HELP_CHAR                    CHAR(1) INIT('?'),
  05 HELP_FIELD_PGM_ID            CHAR(4) INIT('HELP'),
  05 HELP_SCREEN_PGM_ID           CHAR(4) INIT('HELP'),
  05 HELP_TABLE_LIMIT             FIXED BIN(15) INIT(10);
```

hhUPDTA

All programs that update a DL/I database, using the CA Telon.-generated update logic, must use a common update work area defined specifically for CA Telon..

Note: If you use custom code only, you can ignore the update work area.

The update work area defines a single 05-level data item that is at least as large as the largest segment being updated. It is not named explicitly in the non-procedural statements, but it is included automatically in the generated code for each program generated that updates a database segment. You must name its copy library hhUPDTA, where hh is the application HEADER.

The area's definition must be an 05-level item, labeled as follows (for COBOL and PL/I, respectively):

- For COBOL:
05 UPDATE-AREA PIC X(nnnn).
- For PL/I:
05 UPDATE_AREA CHAR(nnnn),

hhPCBs

IMS/DC programs that execute under TSO use a single IMS/DC PSB. You must define the structure of that PSB to every program in your application. To do this, you must use two COPY members: hhPCBs and hhPROC (described in the next topic "hhPROC").

Note: When executing under IMS/DC, PCBs can be either copied in from hhPCBs and hhPROC (as in TSO) or generated automatically based on non-procedural statements that are the input to the generator. In the latter case, hhPCBs and hhPROC need not be defined.

COPY member hhPCBs contains the definition of each PCB that is used by the application. CA Telon automatically copies the COPY member hhPCBs either into the LINKAGE SECTION of a COBOL program or with the other declare (DCL) statements in a PL/I program. The name of each PCB defined here must also be included in the argument list supplied in COPY member hhPROC.

Each PCB must be a 01-level and must have a name that conforms to that specified on the corresponding DBD in data administration (Option 2 of the TDF). The name must be of the form name-PCB, where name is the value supplied on the PCBNAME parameter on the Create/Update PSB, File Group, Update Sensitive Segment, to Update SEGEDIT, or Update DBMS Type screen. If no PCBNAME parameter is specified, name is the value supplied on the DBDNAME parameter on the Update Sensitive Segment, Update Batch Output Field, Update Output, Input, Outin Fields, Update SELECT field, or View Presentation Store screen.

In the name hhPCBs, hh is the application HEADER. COBOL and PL/I examples of COPY member hhPCBs follow.

For COBOL:

```
01 DUMMY-PCB.
    05 FILLER                                PIC X(8) .
01 DUMM2-PCB.
    05 FILLER                                PIC X(8) .
01 HOLD-PCB.
    05 HOLD-DBD-NAME                        PIC X(8) .
    05 FILLER                              PIC X(2) .
    05 HOLD-STATUS-CODE                    PIC X(2) .
    05 FILLER                              PIC X(8) .
    05 HOLD-SEG-NAME-FB                    PIC X(8) .
    05 FILLER                              PIC X(8) .
    05 HOLD-KEY-FB-AREA                    PIC X(8) .
01 HELP-PCB.
    05 HELP-DBD-NAME                        PIC X(8) .
    05 FILLER                              PIC X(2) .
    05 HELP-STATUS-CODE                    PIC X(2) .
    05 FILLER                              PIC X(8) .
    05 HELP-SEG-NAME-FB                    PIC X(8) .
    05 FILLER                              PIC X(8) .
    05 HELP-KEY-FB-AREA                    PIC X(8) .
01 EMPLOYEE-PCB.
    05 EMPLOYEE-DBD-NAME                    PIC X(8) .
    05 FILLER                              PIC X(2) .
    05 EMPLOYEE-STATUS-CODE                PIC X(2) .
    05 FILLER                              PIC X(8) .
    05 EMPLOYEE-SEG-NAME-FB                PIC X(8) .
    05 FILLER                              PIC X(8) .
    05 EMPLOYEE-KEY-FB-AREA                PIC X(8) .
```

For PL/I:

```

DCL DUMMY_PCB_PTR POINTER;
DCL DUMM2_PCB_PTR POINTER;
DCL HOLD_PCB_PTR POINTER;
DCL 1 HOLD_PCB BASED(HOLD_PCB_PTR),
      05 HOLD_DBD_NAME          CHAR(8),
      05 FILL1                  CHAR(2),
      05 HOLD_STATUS_CODE       CHAR(2),
      05 FILL2                  CHAR(8),
      05 HOLD_SEG_NAME_FB       CHAR(8),
      05 FILL3                  CHAR(8),
      05 HOLD_KEY_FB_AREA       CHAR(8);
DCL EMPLOYEE_PCB_PTR POINTER;
DCL 1 EMPLOYEE_PCB BASED(EMPLOYEE_PCB_PTR),
      05 EMPLOYEE_DBD_NAME      CHAR(8),
      05 FILL1                  CHAR(2),
      05 EMPLOYEE_STATUS_CODE   CHAR(2),
      05 FILL2                  CHAR(8),
      05 EMPLOYEE_SEG_NAME_FB   CHAR(8),
      05 FILL3                  CHAR(8),
      05 EMPLOYEE_KEY_FB_AREA   CHAR(8);
DCL HELP_PCB_PTR POINTER;
DCL 1 HELP_PCB BASED(HELP_PCB_PTR),
      05 HELP_DBD_NAME          CHAR(8),
      05 FILL1                  CHAR(2),
      05 HELP_STATUS_CODE       CHAR(2),
      05 FILL2                  CHAR(8),
      05 HELP_SEG_NAME_FB       CHAR(8),
      05 FILL3                  CHAR(8),
      05 HELP_KEY_FB_AREA       CHAR(8);

```

hhPROC

You use COPY members hhPROC and hhPCBs to define an IMS/DC PSB.

Member hhPROC contains an argument list of the PCBs used by the application. This argument list is concatenated onto the end of either the USING statement for COBOL programs or the main PROC statement for PL/I programs. The names of the PCBs must be consistent with the way they are presented in COPY member hhPCBs. Their order must be consistent with the actual PSB.

In the name hhPROC, hh is the application HEADER. Examples of the coding of hhPROCs for both COBOL and PL/I follow.

For COBOL:

```
DUMMY-PCB,  
DUMM2-PCB,  
HOLD-PCB,  
EMPLOYEE-PCB,  
HELP-PCB.
```

For PL/I:

```
DUMMY_PCB,  
DUMM2_PCB,  
HOLD_PCB,  
EMPLOYEE_PCB,  
HELP_PCB;
```

Task 2: Define Database Segment COPY Members

For each database segment to be accessed, a COPY member defining that segment must exist. This COPY member need not be defined specifically for CA Telon, since it can also be used for nonCA Telon applications. It must begin with an 03-level (or lower) item. Generally, the first level should be 05. For PL/I programs, the INCLUDE member must not contain any DCL statements or semicolons.

Task 3: Create a TSO Test PSB

For each application that is tested under TSO, you must define a unique PSB. CA Telon requests the name of the PSB at the beginning of each TSO testing session. The names and the order of the PCBs defined in COPY members hhPCBs and hhPROC must be consistent with those defined in the PSB. You can define a single PSB to be used for your application in both the testing and production phases.

Task 4: Allocate Test Databases

To test using TSO, you must allocate a set of test databases. The procedure for allocating the databases depends upon your installation's standards. In most cases, you must set up a member in a CLIST library that includes TSO ALLOCATE statements for your test databases.

CICS

In the CICS environment, you must define the following three COPY members (INCLUDE members for PL/I) before you can compile a generated program:

- **hhWKAREA** – Defines the work variables used in the application (where hh is the screen HEADER, generally used to identify the application).
- **hhUPDTA** – Applies when you are accessing DL/I files. Defines the work area for generated DL/I update processing. Again, hh is the screen HEADER.
- **File COPY members** – Defines the data files being processed.

hhWKAREA

All programs in a given application use a common application work area, defining the common variables used in that application. This work area includes some standard CA Telon fields, as well as any application-specific fields used during processing. Although you can specify a program work area as well as this application work area, combining all fields that are common in an application into the application work area improves coding efficiency and consistency.

The application work area is not identified explicitly in the non-procedural statements, but is included automatically for each program generated. Its copy library name must be hhWKAREA, where hh is the HEADER for the screen being processed.

The work area can take any form you want. It must include all system reserved fields (described in Chapter 14) as application work area variables, and it can include any number of other (for example, application specific) fields. The area's definition must be an 01-level item (having any user-defined name). The definition of the required application work area fields is shown below for both COBOL and PL/I.

For COBOL:

```
01 WORKAREA.
  05 ERROR-MESSAGE-NOHIT          PIC X(28)
    VALUE '*** NO OPTION SELECTED ***'.
  05 ERROR-MESSAGE-MULTHIT        PIC X(37)
    VALUE '*** MORE THAN ONE OPTION SELECTED ***'.
  05 ERROR-MESSAGE-HIGHLIGHT      PIC X(41)
    VALUE '*** ABOVE HIGHLIGHTED FIELDS IN ERROR ***'.
  05 ERROR-MESSAGE-SELECT-LINE-NO PIC X(28)
    VALUE '*** INVALID LINE NUMBER ***'.
  05 ERROR-MESSAGE-HELP          PIC X(33)
    VALUE '*** RETURN FROM HELP FUNCTION ***'.
  05 ERROR-MESSAGE-HOLD          PIC X(33)
    VALUE '*** RETURN FROM HOLD FUNCTION ***'.
  05 ERROR-MESSAGE-HOLD-ISRT     PIC X(34)
    VALUE '*** CANNOT HOLD CURRENT SCREEN ***'.
  05 MORE-LITERAL                PIC X(7)  VALUE '***MORE'.
  05 ERROR-REQ-CHAR              PIC X      VALUE '*'.
  05 HELP-CHAR                   PIC X      VALUE '?'.
  05 HELP-FIELD-PGM-ID           PIC X(4)  VALUE 'CVH5'.
  05 HELP-SCREEN-PGM-ID          PIC X(4)  VALUE 'CVH5'.
  05 HELP-TABLE-LIMIT            PIC 9(2)  COMP VALUE 10
```

For PL/I:

```
DCL 1 WORKAREA,
  05 ERROR_MESSAGE_NOHIT          CHAR(28)
    INIT('*** NO OPTION SELECTED ***'),
  05 ERROR_MESSAGE_MULTHIT        CHAR(37)
    INIT('*** MORE THAN ONE OPTION SELECTED ***'),
  05 ERROR_MESSAGE_HIGHLIGHT      CHAR(41)
    INIT('*** ABOVE HIGHLIGHTED FIELDS IN ERROR ***'),
  05 ERROR_MESSAGE_SELECT_LINE_NO CHAR(28)
    INIT('*** INVALID LINE NUMBER ***'),
  05 ERROR_MESSAGE_HELP          CHAR(33)
    INIT('*** RETURN FROM HELP FUNCTION ***'),
  05 ERROR_MESSAGE_HOLD          CHAR(33)
    INIT('*** RETURN FROM HOLD FUNCTION ***'),
  05 ERROR_MESSAGE_HOLD_ISRT     CHAR(34)
    INIT('*** CANNOT HOLD CURRENT SCREEN ***'),
  05 MORE_LITERAL                CHAR(7)  INIT('***MORE'),
  05 ERROR_REQ_CHAR              CHAR      INIT('*'),
  05 HELP_CHAR                   CHAR      INIT('?'),
  05 HELP_FIELD_PGM_ID           CHAR(4)  INIT('PVH5'),
  05 HELP_SCREEN_PGM_ID          CHAR(4)  INIT('PVH5'),
  05 HELP_TABLE_LIMIT FIXED BIN(15) INIT(10);
```

hhUPDTA

If your application processes a DL/I file, all programs that update that file, using the update logic generated automatically by CA Telon., must use a common update work area defined specifically for CA Telon..

Note: If you use custom update code only, you need not be concerned with the update work area.

The update work area defines a single 05-level data item that is at least as large as the largest segment being updated. It is not named explicitly in the non-procedural statements, but is included automatically, in the generated code, for each program generated that updates a database segment. Its copy library name must be hhUPDTA, where hh is the HEADER for the screen being processed.

The area's definition must be an 05-level item, labeled as follows (for COBOL and PL/I, respectively):

- For COBOL:
05 UPDATE-AREA PIC X(nnnn) .
- For PL/I:
05 UPDATE_AREA CHAR(nnnn) ,

File COPY Members

For each database segment or VSAM file to be accessed, a COPY member must exist that defines that segment. You need not define this COPY member specifically for CA Telon.; it can be used for nonCA Telon applications as well. It must, however, begin with an 03-level (or lower) item; generally, you use 05-level for the first level. In addition, for PL/I programs, the INCLUDE member must not contain any DCL statements or semicolons.

Transfer Work Area Definition

In CA Telon applications, as with most online applications, data usually has to be transferred between programs in an application and/or from one iteration to another of the same program. Such a data transfer allows one program to remember data entered in another program, to process that data, and finally, to pass it (with any other data required) on to the next program.

In CA Telon, this area is called the transfer work area (XFER), and it is required for all applications. You define it using the XFERWKA field on the Update Program Definition defaults, Create/Update Screen definition, or Create/Update IMS/DC Report Definition screen. CA Telon generates it in the SPA-AREA of the CA Telon program. For the IMS/DC environment, if the generated program is conversational, the transfer work area is located physically in the IMS/DC SPA. If the program is nonconversational, the transfer work area is on a work database (usually keyed by LTERM).

This chapter includes a discussion of the transfer work area, including the following information related to that area:

- Information included
- Required items
- Format definition

Information Included in the Transfer Work Area

Although the actual data in the transfer work area varies by application, it generally includes information such as that listed below:

- **Key fields**—Fields that uniquely identify the data being processed. Often, these key fields correspond to the key fields in the file(s) that contains the data.
- **Program names**—One or more program identifiers that tell the current program from what point it received control and at what point it returns control.
- **Indicators**—Flag values that inform the current program of various activities that have (or have not) been performed and/or the results of those activities.
- **Temporary update data**—Values that are eventually updated in data records, but that are entered using one screen and then updated (probably after some intermediate calculations or checks) in a later program.
- **Loop processing control information**—A key that identifies an item selected from a list of items displayed on one screen, to be processed more fully using another screen. In this case, the key data must be saved in a table in the transfer work area, as described more fully in Chapter 9 under "SEGLOOP".

Required Transfer Work Area Items

As noted above, the XFER area is application-specific and generally includes data items that are user-defined. In some cases, however—specifically when the application is processing a SEGLOOP or using the HELP/HOLD facility—CA Telon.-generated functions use the transfer work area. If you are using these functions, you must define standard fields related to them there. Descriptions of these fields follow.

Required SEGLOOP Fields

If your program uses SEGLOOP processing, its transfer work area must include the SEGLOOP last-line number, and it can also include the SEGLOOP save-key table, select-key-save value, and page area, depending on the activity being performed.

SEGLOOP last-line number

SEGLOOP last-line number, required for all SEGLOOP processing, holds the number of iterations of the loop displayed on the screen and is used for control during input processing. For example, if the screen holds up to 12 iterations of data but only seven are written out, this field is set (automatically) to seven.

You must define the field exactly as shown below, except that the item level number can vary. It can appear anywhere in the transfer work area.

- For COBOL:
05 LAST-LINENO PIC 9(2).
- For PL/I:
05 LAST_LINENO CHAR '99',

SEGLOOP page area

SEGLOOP page area is a work area that is required when an application pages through a looping screen; that is, when PAGE=Y on the Create/Update File SEGLOOP screen for any of the application's screens. While the specific paging work fields are defined automatically (in this area), this area is necessary to provide adequate space for them in the transfer work area. When it is included, the SEGLOOP page area field must be the last field defined in the XFER area.

Note: You can calculate the size of the page area only if the keys are unique.

You must define the field exactly as shown below.

Note: The item level number must be 05 for COBOL.

- For COBOL:
05 PAGE-AREA-START PIC X(nnn).
- For PL/I:
05 PAGE_AREA_START CHAR(nnn),

Where nnn is the size of the maximum page area used by all SEGLOOP paging screens in the application.

You must calculate the maximum page area for each SEGLOOP screen with paging and use the largest value. (When calculating this value, do not be concerned with those screens that use SEGLOOP processing but not paging.) Calculate the page area as follows:

$SIZE = 5 + ((\text{save-count} + 2) * \text{key-length})$
+ search/browse-field-length

- Save-count—The PAGESAV value specified on the Create/Update File SEGLOOP screen.
- Key-length—The length of the segment (for DL/I) or record (for VSAM) key field used in (paging) SEGLOOP processing. Specifically, it is the value of the field named by the KEYLEN parameters in the BROWSE Use I/O in the data group.
- Search/browse-field-length—The size of the search field specified using the following SEGLOOP related parameters:
 - SCHFLDC
 - SCHFLDI
 - SCHFLDL use with DL/I files only or with start-browse key which is specified using the SEGLOOP statement's STBRKEY used in (paging) SEGLOOP processing.

In the absence of a search or browse field, use a zero (0) value for this item.

Note: The SCHFLDx fields are only available for carry-over from pre-2.x releases of CA Telon. These fields only appear when they are already defined in a program imported to the TDF.

For example, suppose the PAGESAV is 10, the KEYLEN is 4, and the search field is 8. You would calculate the page area size as follows:

$5 + ((10+2) * 4) + 8 = 61$

Assuming this is the largest calculated page-area size for all paging SEGLOOP screens in the application, you define the page area as PIC X(61) for COBOL or CHAR(61) for PL/I. Generally, you should leave some room for future use. A size of 80 is more appropriate here.

SEGLOOP save-key table

You must code a save-key table when the SAVEKEY parameter is specified on the Create/Update Table SEGLOOP or the Create/Update File SEGLOOP screen. It is a table used to save the key or other identifying data for each iteration of the list displayed on the screen. The table's definition must correspond to the format of the key data itself. Each set of key values should repeat as many times as the loop displays on a single screen. For more information on SEGLOOP processing, see Chapter 9, "Processing Flow."

The following example codes a save-key table for a SEGLOOP process with a 9-byte key and 13 iterations of a loop displayed on a screen.

- For COBOL:

```
05 XFER-SEGLOOP-TABLE OCCURS 13  
10 XFER-TABLE-MSG-KEY PIC X(9).
```

- For PL/I:

```
05 XFER_SEGLOOP_TABLE(13),  
10 XFER_TABLE_MSG_KEY CHAR(9);
```

SELECT Key-Save Value

You must define a SELECT key-save value when the SELKEY parameter is specified on an Update SELECT Field screen. If you use CA Telon's automatic line-number tie-in to a selected field, the SELECT key-save value is the field in which the key values (saved in the save-key table described above) are stored for a selected screen line item. Through this key-save value, you can access detailed information for the selected line item from a subsequent screen. For more information, see List Processing in the chapter "Processing Flow."

Required HELP/HOLD Fields

When a program uses the CA Telon HELP or HOLD facility, its transferwork area must include the fields described below:

- For HOLD processing, it must include the XFER-HOLD-INDICATOR, a one-byte character field used to indicate to CA Telon that the current program was resumed following a hold

- For HELP processing, it must include:
 - HELP-CURR-MSG-COUNT – A numeric field used to store the message number being displayed on the HELP screen
 - HELP-MSG-COUNT – A numeric field used to store the number of message keys provided when each HELP request is made
 - HELP-MSG-NAME – An array of keys to the HELP database. The number of entries in this array is defined using the HELP-TABLE-LIMIT field in the hhWKAREA variable storage area, described earlier in this chapter. The length of each entry in the table must be at least as large as the length of the message name specified by the HELPMMSG parameter on the Update HELPMMSG parameter; Update Output, Input, Outin Fields; or Update SELECT Fields screen.

XFER Format Definition

Identify the transfer work area by using the XFERWKA field of the Update Program Definition, Create/Update Screen Definition, or Create/Update IMS/DC Report Definition screen. This parameter names one or more COPY members whose contents, together, define the XFER area. You can name up to 20 COPY members in the XFERWKA parameter. When CA Telon generates source code, it includes the transfer work area as an 02-level item in the 01-level SPA-AREA item. The work area can take any form you want, but if the application is performing SEGLOOP processing, it must include the fields described previously under "Required Transfer Work Area Items". (If the XFER area includes the SEGLOOP page area discussed above, that area must be defined last.)

Make sure that each item in the transfer work area is carefully documented, either in comments in the COPY member or in a separate section that is copied into each program for the application. Because the transfer work area transcends screen-level design, it involves some of the most complex system design for any given application. Therefore, be sure to document it thoroughly in an identical manner in each application program.

The area's definitions must begin with a 03-level (or higher-numbered level) item because, as noted above, CA Telon generates the XFERWKA copy statement at the 02-level. In addition, for PL/I programs, the member must not contain any DCL statements or semicolons.

Chapter 13: Application System Generator

The CA Telon Application System Generator generates a COBOL or PL/I application program. The generated code can process a screen, produce a batch program, or act as an online driver/controller. The input to the Application System Generator is called a screen definition, a report definition, a driver definition (depending upon the processing to be done by the program), or a batch definition. These definitions are created from TDF parameters and procedural custom code, as described below:

- TDF parameters provide the information that CA Telon uses to generate source code, protocol, control blocks, screen-field mapping, editing logic, data definitions, and data I/O. CA Telon can generate a complete COBOL or PL/I program from TDF parameters alone. Before generation, TDF parameters are converted into CA Telon source which can be considered non-procedural statements. This chapter lists the non-procedural statements and their associated parameters. The associated parameters are the parameters that you defined earlier on the various TDF screens.
- Procedural custom code adds logic to CA Telon programs, in addition to the logic that CA Telon generates automatically. In most instances, the custom code includes logical (i.e., decision-based IF/THEN/ELSE) processing. Each block of custom code is requested from the non-procedural statements, serves a specific predefined purpose, and is placed into the CA Telon-generated program at the area that you specify. (In each case, the exact purpose of the custom code is determined by the keyword through which the custom code is referenced in the non-procedural statements.) The custom code is either defined in a copy library or included within the definition.

This chapter discusses the following topics:

- Components of a definition
- Statement parameter lists
- Procedural custom code

Components of a Definition

The eight categories of CA Telon source statements are as follows:

- Data search criteria
- Data sorting
- Definition statements
- Data access
- Definition literals
- Definition variables
- Consistency edits
- Environment statements

Specify these statements by initiating and processing various TDF screens. To cross-reference the screen on which the statements and parameters appear, see "Statement Parameter Lists" later in this chapter.

Data Search Criteria

Data search criteria defines DL/I or EXEC DLI qualification options.

DLIDSC Statement

This statement defines DL/I or EXEC DLI qualification options. For DL/I, each DLIDSC causes the generation of an SSA. For EXEC DLI, qualification parameters impact the generated data access (and possibly working storage).

Data Sorting, Matching, or Merging

CA Telon batch allows mainline sorts, matches, and merges. Data sorting statements define these sorts.

AUTOEXEC Statements

The Generator will process macro statements for generating MATCHM (match master), MATCHT (match transaction), and MERGENN (MERGE01 through MERGE20) AUTOEXEC calls.

MATCH statement

Export will create a MATCH statement that the Generator will use to produce a MATCH program.

MATCHKEY statement

The Generator uses the MATCHKEY statement to produce matching logic for each match key in a MATCH program.

MERGE statement

The Export utility produces the MERGE statement as an intermediate step to Generating a MERGE program.

MERGEGRP statement

The Generator uses the MERGEGRP statement to produce MERGE logic for each key group in a MERGE program.

MERGEKEY statement

The Generator uses the MERGEKEY statement to produce MERGE logic for each key in a MERGE program.

SORT statement

The Export utility produces and the Generator processes one SORT statement for each sort in a program (either mainline or user-defined) and one SORTKEY statement for each sort key defined to the sort whose CA Telon statement it follows.

The data for mainline and user-defined sorts is stored on two different database segments with identical structures.

SORTKEY statement

The SORTKEY statement is generated for each sort key specified for a mainline or user-defined sort.

Definition Statements

Definition statements provide program specific information. The following are definition statements:

BATCH statement

This statement specifies the characteristics of a CA Telon batch program.

DRIVER statement

This statement indicates that the program is an online driver program in a statically-linked system. This statement also defines the online driver program's characteristics.

NONTERM Statement

The NONTERM statement drives nonterminal program definitions. It uses many of the parameters currently used in the BATCH statement.

REPORT statement

This statement indicates that any output should be routed to an IMS/DC printer (rather than to a terminal screen). This statement also defines characteristics of the print subroutine that is built for this type of definition.

SCREEN Statement

This statement indicates that the program processes a screen. It also specifies screen characteristics, including:

- Application- and screen-identifying information
- Control data
- Program remarks
- Cursor positioning
- The program to which control passes from this screen
- Branching logic (based on PF-key processing)
- The name of custom code blocks within parameters
- Work areas and transfer work areas via custom code block names
- The indication of whether to use the system HELP or HOLD facilities

STORED Statement

This statement specifies the characteristics of a CA Telon Stored Procedure program.

TELON statement

This statement identifies the version of CA Telon under which code is generated.

Data Access

These statements define each DL/I database, and identify DB2 tables, columns, rows, CICS journals, CICS queues, etc.

CJOURNAL statement

The CJOURNAL statement supports the definition of CICS journals.

CQUEUE statement

The CQUEUE statement supports the definition of temporary storage and transient data queues.

DATABAS statement

This statement defines each DL/I database that the program uses. The DATABAS statement includes the following:

- Name of the DBD
- PCB-name prefix
- Key length of the longest database path to be accessed
- Secondary index name

DATA SET statement

This statement defines the VSAM or sequential file used by the generated program (if any). The DATA SET statement includes the name of the data set and the types of access for the data set being defined.

DB2 Statement

This statement identifies the DB2 table to which the user exec I/O pertains, which follows the DB2 statement.

DCLCOL statement

This statement defines the columns in the TLNROW statement preceding this statement.

JOINCOL statement

This statement defines the column names (and the corresponding table) to be used to generate the JOIN criterion.

ROW statement

This statement identifies the DB2 table or TLNJOIN to which the user exec I/O pertains, which follows the ROW statement.

SEGMENT RECORD statement

This statement defines each segment/record that the program uses. This statement indicates:

- Key information
- Copy member information for the segment/record

This statement also specifies other information related to the generation of calls for the segment/record.

TABLE statement

This statement describes the SQL table/view being referenced.

TLNJOIN statement

This statement defines the SQL tables/views to be referenced in a JOIN.

TLNROW statement

This statement permits multiple CA Telon views of a DB2 table/view. Each TLNROW permits different specifications of the CA Telon-defined column characteristics for each column in a DB2 table.

TPPCB Statement

This statement identifies any teleprocessing PCBs that the program uses.

Data access I/O statements

These statements are as follows: BROWSE, CREATE, DELETE, ERASE, HOLD, INREAD, JOURNAL, MATCHX, MERGE#, OIREAD, OUTREAD, READ, READNEXT, REPLACE, TRANSACT, UPDATE, and WORKSPA.

These statements request that, at program generation time, CA Telon create a program paragraph (or procedure for PL/I) that performs data I/O. These statements are called *user exec I/O statements*. Alternately, the I/O can be positioned within the automatically executed sections of the CA Telon program. This type of data I/O is called an auto exec I/O statement.

Definition Literals

List of the literal values that are presented on the screen or report.

FIELD Statement

This statement specifies the characteristics of each literal field on the screen or report, including the:

- Field's usage (literal)
- Field's position and content
- Screen attributes (i.e., protection, intensity, highlighting, and color)

Definition Variables

This topic lists variable type fields that are presented on the screen or report.

FIELD statement

This statement specifies the characteristics of each field on the screen, including the following:

- Field's usage (output, input, outin, or select)
- Field's position and length
- Screen attributes (i.e., protection, intensity, highlighting, and color)
- Source (on output) or target (on input) storage area
- Field edit routine to be executed for the field variable name
- Formatting, conversion, and acceptable values or ranges for input fields
- Name of the associated HELP text key, if any

GROUP statement

This statement identifies the beginning of a logical item or entry of one or more lines to be printed in a batch report.

GROUPEND statement

This statement identifies the end of a group in a batch program.

SEGLOOP and SEGEND statements

These statements identify, respectively, the beginning and end of loop processing. (These statements can be included in both screen and report definitions.) The SEGLOOP statement also defines the characteristics of the loop, including:

- Whether the loop processes input, output, or outin fields
- Line spacing (down the page)
- Column spacing (across the page)
- The use of paging
- Custom code block names
- Database search-key information.
- Information necessary to maintain an array of records included in the loop (key values, array index, first-key value, and so on)

Consistency Edits

Perform consistency checking to ensure that fields are consistent with other data in the program.

SEGEDIT statement

This statement requests that, at program generation time, CA Telon creates code to access a database segment/record in order to verify its existence and handle resulting errors.

SRC statement

This statement inserts a line of COBOL or PL/I custom code in Consistency Edit statements SEGEDIT and XFEDIT.

XFEDIT Statement

This statement requests that, at program generation time, CA Telon code to perform a cross field validation and to handle any resulting errors.

Environment Statements

Environment statements define information about the environment in which the program is generated.

BATCHPGM statement

This statement requests the generation of a CA Telon batch program.

CICSBMS statement

This statement requests the generation of a BMS map.

CICSPGM statement

This statement requests the generation of a CICS program and defines its characteristics.

DLIPSB statement

This statement requests the generation of a PSB for screens that access DL/I files.

IMSDRV statement

This statement requests a generation of an online IMS driver program and defines its characteristics.

IMSMFS statement

This statement requests a generation of the IMS MFS control blocks for a screen.

IMSPGM statement

This statement requests a generation of an IMS program and defines its characteristics.

IMSPSB statement

This statement requests a generation of an IMS PSB.

PLIXOPT statement

For applications written in PL/I, this statement overrides specific PL/I defaults defined for the application.

SPPGM Statement

This statement requests a generation of a Stored procedure program and defines its characteristics.

TSOPGM statement

This statement requests generation of a TSO online program and defines its characteristics.

Statement Parameter Lists

The charts in this subsection list the CA Telon high-level statements and their associated parameters. These are statements generated by the TDF as an intermediate step in the creation of a CA Telon-generated COBOL or PL/I program.

The lists are alphabetical by statement and by parameter within each statement. Positional parameters are shown in lower case, since they are user-defined. There are five columns on each list. They are as follows:

1. The name of the parameter.
2. The environment in which the parameter is used. The values are as follows:
 - B—Batch
 - C—CICS
 - I—IMS
 - T—TSO
 - S—Stored procedure
3. The maximum size (in bytes) of the parameter.
4. Any special format and/or description of the parameter.
5. The number of the screen or screens where you may enter or change this parameter in the TDF. These numbers are cross referenced to both the TDF Organization Chart and the TDF Screens listed earlier in this guide.

CA Telon source statements are created by a CA Telon *export* and are the input to the CA Telon generator. Many installations save the CA Telon source as the "true source" reflecting the generated code.

CA Telon source may also be input to a CA Telon *import* to add a program back to the CA Telon TDF for maintenance. A programmer may see the CA Telon source. The explanations given here are to help the understanding of the connection between the generated code and the TDF design steps. Programmers should not modify CA Telon source statements. This would make the generated code different than the TDF design and furthermore may make an import back to the TDF impossible, thus *endangering* future maintenance.

BATCH Statement

This statement specifies the characteristics of a CA Telon batch program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
APPLID	B	7	Application identifier	B110
CMPLOPT	B	253	(parameter1 [,parameter2,] [parametern])	B110
COBFCPY	B	16	(mbrname1.mbrname2) Each <i>mbrname</i> 8 bytes max.	B110
CRTDATE	B	8	YY/MM/DD	-
DESC	B	40	Batch definition description	B100, B110
GETTRAN	B	8	<i>mbrname</i>	B110
HEADER	B	5	Program header	B100
ID	B	5	Program identifier	B100
IDENTIF	B	8	<i>mbrname</i>	B110
INIT1	B	8	<i>mbrname</i>	B110
INIT2	B	8	<i>mbrname</i>	B110
LANG	B	3	COB PL/I	B110
PARMS	B	46	(parm-lth1[,parm-lth2] ... [,parm-lthn].) Each parm-lth 2 bytes long	B110 B110
PGMCUST	B	240	(section1,mbrname1 [,section2,mbrname2] ... [section,mbrnamen]) (max of 50)	B110
PRCTRAN	B	8	<i>mbrname</i>	B110
PROCEDR	B	8	<i>mbrname</i>	B110
REMARKS	B	8	<i>mbrname</i>	B110

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
RPTDEST	B	8	<i>ddname</i>	B110
SECTION	B	240	(mbrname1[,mbrname2] ... [,mbrnamen]) (max of 35)	B110
SIZE	B	7	(lll,ccc) lll = line position ccc = column position.	B110
STRUCTRE	B	5	SORT = Mainline sort, MATCH = Match, MERGE = Merge (No STRUCTRE parameter means that the program's structure is Standard.)	B110
TERM	B	8	<i>mbrname</i>	B110
UPDDATE	B	8	YY/MM/DD	-
UPDTIME	B	4	HH:MM	-
UPDUSER	B	8	User ID that last updated the program in the TDF.	-
WKAREA	B	240	(mbrname1[,mbrname2] ... [,mbrnamen])	B110
XFERWKA	B	8	<i>membrane</i>	B110

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

BATCHPGM Statement

This statement requests the generation of a CA Telon batch program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DBMS	B	8	DBMS Type: VSAM DL/I DB2 SEQ DL/I EXEC DL/I EXECCLI	F113, S165
DLIWGHT	B	1	Y N Y = Automatic DL/I call weighting	B110
GENPCBS	B	1	Y N Y = generate PCB masks automatically N = include masks coded in LNKCOPY and USGCOPY	B168

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
members				
LNKCOPY	B	8	<i>mbrname</i>	B168
PGMNAME	B	8	Program-name Overrides default name	B168
TRACE	B	1	Y N Y = generate trace variables	B168
USGCOPY	B	8	<i>mbrname</i>	B168

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

BROWSE Statement

The following list contains parameters that are unique to the BROWSE statement. See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
GENKEYL	C	30	<i>hvname</i> n1 Generic key length	S146
LTHLBL	C	30	Overrides default generated COBOL or PL/I queue LTHOPT variable or record-level LTHLBL	S14Q
OPCODE	CIT	5	Ssa-opcode GTEQ EQUAL	S147, S145 S146
RECLTH	C	60	<i>hvname</i> n1 Record-length	S146
WHERE	CITS	240	(DB2 only) Specifies the WHERE condition to be used in the generated USER I/O	S147, S146

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

CICSBMS Statement

This statement requests the generation of a BMS map.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
BMSMAP	C	8	BMS map name override	S165
LINEOPT	C	1	1 2 3 1 - Use CA Telon line optimizing subroutine 2 - Simulate subroutine optimization in procedural custom code 3 - Do not generate line optimizing code	S165

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

CICSPGM Statement

The CICSPGM CA Telon statement is used for both the CICS screen or Nonterminal programs.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
BMS	C	1	Y N Y = create a BMS map N = use CA Telon's mapping	F113, S165
BMSMAP	C	8	BMS map name override	S165
DBMS	C	8	DBMS Type: VSAM DL/I DB2 SEQ DLI EXECCLI	F113, S165
ENV	C	8	CICS environment MCSCICS = CICS Client/Server	U100
GENPCBS	C	1	Y N Y = generate PCB masks automatically N = include masks coded in LNKCOPY and USGCOPY members	F113, S165
IOASTG	C	1	A S A = Auto S = Static (COBOL only)	F113, S165
LINEOPT	C	1	1 2 3	F113, S165

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			1 = use CA Telon line optimizing subroutine 2 = simulate subroutine optimization in procedural custom code 3 = Do not generate line optimizing code	
LNKCOPY	C	8	<i>mbrname</i>	F113, S165
PSBNAME	C	8	<i>psbname</i> DL/I only - 8 bytes per name max.	F113, S165
PSBSCHD	C	1	Y N Y = generate PSB schedule automatically N = do not generate PSB schedule. PSB must be scheduled in custom code	F113, S165
SPASIZE	C	5	Size of the DFH COMMAREA	F113, S165
SPASTG	C	1	A S A = Auto S = Static	F113, S165
TRACE	C	1	Y N Y = generate trace variables	F113, S165
TRANCDE	C	4	Name of the CICS transaction code if different from the default	S165
TPBSTG	C	1	A S A = Auto S = Static COBOL only	F113, S165
USGCOPY	C	16	(mbrname1,mbrname2) mbrname1 = BLL-POINTER-LIST mbrname2 = Q-100-CICS-INIT COBOL only - 8 bytes per name max.	F113, S165

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

CJOURNAL Statement

The CJOURNAL statement supports the definition of CICS journals.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
NAME	C	8	CICS journal name (CA Telon use only)	D11J, S13J
LRECL	C	5	Identifies maximum length of journal record	D11J, S13J
JFILEID	C	2	CICS Journal ID (01 through 99)	D11J, S13J, S14J
JTYPEID	C	2	Journal type ID (valid for JOURNALS only)	D11J, S13J S146D

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

CQUEUE Statement

The CQUEUE statement supports the definition of temporary storage and transient data queues.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LRECL	C	5	Identifies maximum length of queue record	D11Q, S13Q
MAINAUX	C	1	Identifies Temporary Storage queue Storage as main or auxiliary (A=AUX , M=MAIN).	D11Q, S13Q S146C
NAME	C	8	CICS queue name	D11Q, S13Q
QUETYPE	C	2	Identifies type of queue (TS=Temporary storage ; TD=Transient Data)	D11Q, S13Q
SYSID	C	4	CICS system identifier	D11Q, S13Q, S14Q

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

CREATE Statement

The following list contains parameters that are unique to the CREATE statement. See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LTHLBL	C	30	Overrides default generated COBOL or PL/I queue LTHOPT variable or record-level LTHLBL	S14Q
GRPISRT	BCITS	240	Group insert specification	
RECLTH	BC	60	<i>hvname</i> n1 record-length	S146
TPPARMS	BCIT	240	(<i>hvname1</i> [, <i>hvname2</i>] ... [, <i>hvnamen</i>]) parm1, parm2	S149

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DATA ACCESS I/O Statement

- BROWSE (BR)
- CREATE (CR)
- DELETE (DE)
- ERASE
- HOLD
- INREAD (IN)
- JOURNAL
- MATCHX
- MERGE#
- OIREAD (OI)
- OUTREAD (OU)
- READ (RE)
- READNEXT (RN)
- REPLACE
- SPBROWSE
- SPRDNEXT

- SPTRNACT
- TRANSACT
- UPDATE (UP)
- WORKSPA

The following list contains parameters that are common to all data access I/O statements. To obtain a list of unique parameters, see each individual statement throughout this chapter.

Note: This list has one addition column, ACCESS:SVD2, which shows the types of data access available.

The following statements are not valid for BATCH:

- BROWSE
- HOLD
- INREAD
- JOURNAL
- OIREAD
- OUTREAD
- SPBROWSE
- SPRDNEXT
- WORKSPA

The MATCHX, MERGE#, SPTRNACT, and TRANSACT statements are not valid for Online (CICS, IMS, TSO). Browse is not valid for TD queues; JOURNAL is only valid for journals (in CICS programs).

Parameter	Env *BCITS	Size Bytes	Access: SVD2**	Format/description	TDF Screens
ALTCURS	BCITS	32	2	Alternate cursor that points at another data access request and used in this data access request.	S244
ALTSSA	BCIT	1	D		
CASTAS	BCITS	240	2	Comma-separated list of user data type specifications for SENCOLS columns.	S184
CASTAS2	BCITS	240	2	Comma-separated list of user data type specifications for SENCOLS columns.	S184
CASTCOL	BCITS	240	2	Comma-separated list of pointers to SENCOLS/SENCOLS2 columns for	S184

Parameter	Env *BCITS	Size Bytes	Access: SVD2**	Format/description	TDF Screens
				CASTAS/CASTAS2 list entries.	
CASTCOL2	BCITS	240	2	Comma-separated list of pointers to SENCOLS/SENCOLS2 columns for CASTAS/CASTAS2 list entries.	S184
CASTCONT	BCITS	1	2	Flag indicating CASTAS2 contains information.	S184
CCOLCONT	BCITS	1	2	Flag indicating CASTCOL2 contains information.	S184
CMDCODE	BCIT	4	D	(DL/I only) Overrides the default command code as specified in the DLIDSC. Referenced by this user exec for the lowest level segment only.	S145
CONCATK	BCIT	1	D	(DL/I only) Y N Y = a concatenated key is used to qualify an I/O request N = a qualification statement is used to qualify an I/O request	S145
CPYCALL	BCITS	8	2	(DB2 only) Copy member name	S147
CPYINIT	BCITS	8	2	(DB2 only) Copy member name	S147
CPYKEY	BCITS	8	2	(DB2 only) Copy member name	S147
CPYTERM	BCITS	8	2	(DB2 only) Copy member name	S147
CURRENT	BCIT	8	D	(DL/I only) Specifies the segment above which position to maintain for all segment levels for this I/O request.	S145
DELETE	BCITS	1	2	Y N Y = causes an extra USER I/O procedure or paragraph to be generated	S147
DSCREF	BCIT	240	D	(DL/I only) Specifies the DLIDSC statements to use in building the qualification statements or SSA's for this USER I/O request.	S145
FDREC	BC	30	SV	Host variable name of 01 COBOL record definition	S146
FETCH	BCITS	1	2	(DB2 only) Y N Y = generates the FOR FETCH ONLY	S244

Parameter	Env *BCITS	Size Bytes	Access: SVD2**	Format/description	TDF Screens
				clause in DECLARE CURSOR.	
FTCHOPT	BCITS	66	2	Composite of four comma-separated subparameters: FETCH orientation, number of rows to be fetched, row number to be fetched, and FETCH sensitivity. Number of rows and row number can be numeric or a host variable.	S244
FUNC		8	VD	Access code (for example, GU, GHU)	S145, S146, S149, S185
GENKEYL	BC	30	V	(VSAM only) Specifies the length of the generic record used to access the VSAM record. Can be an integer or a variable.	S136, S146
GROUPBY	BCITS	120	2	column name(s)	S147, S144
GRPISRT	BCITS	240	2	Column name(s) for mass insert	S147
GRPISRT2	BCITS	240	2	Column name(s) for mass insert	S147
HAVING	BCITS	120		(DB2 only) Having condition	S147
HOLDCUR	BCITS	1	2	(DB2 only) Y N Y = generates the HOLD clause in DECLARE CURSOR.	S147
IGNORE	BCITS	60	SVD2	(status-code1[, status-code2]... [,status-coden])	S125, S147, S145 S185, S146
IOAREA	BCITS	60	SVD2	<i>hvname</i> , data area name	S125, S147, S145 S185, S146
ISOLATE	BCITS	1	2	Isolation clause. Values are: C U R L S K. C = generates CS; U = generates UR; R = generates RR; L = generates RR KEEP UPDATE LOCKS; S = generates RS; K = generates RS KEEP UPDATE LOCKS	S244
ITEM	C	1	Q	Generate ITEM option N : Y (BR, CR, DE, IN, OI, OU, RE, RN, UP)	S14Q
ITEMLBL	C	30	Q	Overrides generated ITEM variable name. (BR, CR, DE, IN, OI, OU, RE, RN, UP)	S14Q

Parameter	Env *BCITS	Size Bytes	Access: SVD2**	Format/description	TDF Screens
JOINOPT	BCITS	1	2	Join option. Values are: FULL LEFT RIGHT INNER. FULL = generates FULL OUTER JOIN; LEFT = generates LEFT OUTER JOIN; RIGHT = generates RIGHT OUTER JOIN; INNER = generates INNER JOIN	S147
KEY	BCITS	60	D2	<i>hvname</i> key field value	S147, S146, S144, Z102
KEY2	BCITS	60	2	<i>hvname</i> key field value	S147, S144
KEYCOLS	BCITS	240	2	column names	S147, S144
KEYCONT	BCITS	1	2	Flag indicating KEY2 contains information.	S147, S144
label	BCITS	8	SVD2 QUJ	U-100 section name ex. U-100-label-segname	S125
LOCKED	BCIT	8	D	Reserves the specified segment (and any lower level segments in the hierarchy) for the exclusive use by the program, regardless of whether the program updates it. LOCKED tells DL/I to keep a segment from being altered by other programs until the next sync point (equivalent to the "Q" command code).	S145
LTHOPT	C	1	QJ	Y/N Y = generator of 1/0 with CICS LENGTH option	S13J, S13Q S14J, S14Q
MERGSEQ+	B	2	SVD2	(MERGE only) Indicates hierarchical position of merge file.	S125
NITMLBL	C	30	Q	Override generated NUMITEM variable name (BR, CR, DE, IN, OI, OU, RE, RN, UP)	S14Q
NOSUSP	C	1	QUJ	Generates NOSUSPEND option (N Y) (CR, IN, OI, OU, RE, RN)	S14Q
NUMITEM	C	1	Q	Generates NUMITEM option N Y (BR, CR, DE, IN, OI, OU, RE, RN, UP)	S14Q
OPCODE	BCIT	5	VD	GTEQ Equal	S145, S146, S147
OPTLIST	C	60	V	(option1[,option2] ... [,optionn]) values = RRN, SEGSET, RBA,	S146

Parameter	Env *BCITS	Size Bytes	Access: SVD2**	Format/description	TDF Screens
				SEGSETALL, SYSID, MASSINSERT, DEBKEY, DEBREC	
OPTROWS	BCITS	5	2	Number of rows for optimization. Default is to not generate the OPTIMIZE for nnnn rows clause.	S244
ORDERBY	BCITS	120	2	Column name(s)	S147, S144
PARENTG	BCIT	8	D	Specifies the segment at which to override parentage normally set by DL/I (EXEC DLI equivalent to the "P" command code).	S145
PATH	BCIT	8	D	Specifies the highest level segment to be included in path processing for this user exec.	S145
RECLTH	C	60	V	Either an integer or a variable defines the length of the record.	S146
RESLTCC	BCITS	8	2	The name of the custom code member used at the entry point where a stored procedure is being called.	S147
RESLTNR	BCITS	2	2	The number of the result set created by a called stored procedure.	S147
RESLTPR	BCITS	8	2	The name of the stored procedure being called to produce the result set referred to by the RESLTNR field.	S147
RETNCUR	BCITS	1	2	Used in a DECLARE CURSOR statement. Values are: Y N C. Y = generates WITH RETURN; N = generates WITHOUT RETURN; C = generates WITH RETURN TO CALLER	S147
SEGKEY	BC	60	V	Defines the variable name that contains the segment key.	S136
SEGLTH	BC	60	V	Either an align or a variable defining the length of the key.	S136
<i>segname</i>	BCITS	8	SVD2 QUJ	Name of the segment or record	S125
SENCOLS	BCITS	240	2	Column name(s) and/or DB2 function(s)	S147
SENCOLS2	BCITS	240	2	Column name(s) and/or DB2 function(s)	S147
SENCONT	BCITS	1	2	Flag indicating SENCOLS2 contains	S147

Parameter	Env *BCITS	Size Bytes	Access: SVD2**	Format/description	TDF Screens
information.					
SET	C	1	QU	Generates SET option N Y; (N=generate INTO) (CR, IN, OI, OU, RE, RN)	S14Q
SETLBL	C	30	QU	Overrides generated SETLBL variable name. (CR, IN, OI, OU, RE, RN)	S14Q
SSALIST	BCIT	240	D	(hvname1[,hvname2] ... [hvnamen]) first SSA, second SSA,...	S145
TYPE++	B	1	SVD2	(MATCH only) Indicates whether MASTER (M) or TRANSACTION (T) file	S125
UPDATE	BCITS	1	2	Y N Y = generates an extra USER I/O procedure or paragraph	S147
UPDCOLS	BCITS	240	2	Column name(s) of columns to be updated	S144
WHERE	BCITS	240	2	Column name(s)	S147, S144
WHERE2	BCITS	240	2	Column name(s)	S144

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

** S = Sequential; V = VSAM; D = DL/I; 2 = SQL; Q = TS Queue; U = TD Queue;
J = Journal

+shown as 2-digit number affixed to MERGE DATA ACCESS I/O (e.g., MERGE04)

++shown as 1-character code affixed to MATCH DATA ACCESS I/O (e.g.,
MATCHM)

DATABAS Statement

This statement defines each DL/I database that the program uses. The DATABAS statement includes the following:

- Name of the DBD
- PCB-name prefix
- Key length of the longest database path to be accessed
- Secondary index name

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DBDNAME	BCIT	8	<i>dbdname</i>	D100, S125
KEYLEN	BCIT	3	Key length	D211, S135, D114
PCBNAME	BCIT	12	<i>pcbname</i>	D211, S127
PROCOPT	BIT	4	IMS processing options	D211
PROCSEQ	BCIT	8	<i>dbname</i> Secondary index	D211, S127
TYPE	BIT	7	DB2 DL/I DATACOM GSAM IDMS SEQ TPPCB VSAM REF	D111, S127

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DATASET Statement

This statement defines the VSAM or sequential file used by the generated program (if any). The DATASET statement includes the name of the data set and the types of access for the data set being defined.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ACCESS	BC	11	For COBOL or PL/I online: SEQ VSAM, {KSDS RRDS ESDS}, For COBOL batch: SEQ VSAM, {KSDS RRDS ESDS} [DYN RAN SEQ]	D114, S136

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			For PL/I batch: SEQ VSAM, {KSDS RRDS ESDS}, {SEQ DIR}	
BLKSIZE	B	5	Blocking factor size	D114, S136
IGNOPEN	B	1	Y N Ignore ALREADYOPEN condition on OPEN	S136
IGNEMPT	B	1	Y N Ignore EMPTYFILE condition on OPEN	S136
IGNCLOSE	B	1	Y N Ignore ALREADYCLOSE condition on CLOSE	S136
INDEXOF	B	7	<i>dbname</i> Primary data set name COBOL only	D114, S136
LRECL	B	10	Rec-length (mini-rec-length, max-rec-length) Five bytes each length	D114, S136
NAME	BC	8	<i>ddname</i> Data set name	D100, S125
OPEN	B	6	Type of access COBOL = INPUT OUTPUT IO EXTEND PL/I = INPUT OUTPUT EXTEND	D114, S136

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DB2 Statement

This statement identifies the DB2 table to which the user exec I/I pertains, which follows the DB2 statement.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
TBLNAME	BCITS	18	Table name	D100, D411, D141, S125, S127
TBLQUAL	BCITS	8	Qualifier for table name	D100, D411,

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
				D141, S125, S127
SYNONYM	BCITS	1	Y N Y = use DB2 synonyms (table name is not qualified) N = table is qualified	D141, S127

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DCLCOL Statement

This statement defines the columns in the TLNROW statement preceding this statement.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ACCESS	BCITS	1	Y N Y = column will be accessed by the generated EXEC SQL statement.	D141
ALIAS	BCITS	30	<i>hvname</i> Used in place of the DCLGEN <i>hvname</i> for the column	D141
DCLHOST	BCITS		COBOL <i>hvname</i> of the column (not maintained by the user)	Not on TDF
DEC	BCITS	2	Specifies number of digits to the right of decimal point in a variable of DECIMAL data type. Ranges from 0 to length of variable.	D141
JOINLBL	BCITS		Specifies the correlation name of the DB2 table associated with the column (for TLNJOIN)	D142, D143, D144
KEY	BCITS	2	Y N number KEY = Y declares the column to be a "key" column Number = generated WHERE condition will include this column in the specified number order	D141, D144
LTH	BCITS	4	Length of TYPE parameter. Use only for DECIMAL, CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types.	D141, D144
NAME	BCITS	18	Column name	D141, D144

NOTNULL	BCITS	1	Y N N = null is acceptable. A notnull indicator will be generated with any USER I/O reference to the column as a host variable.	D141, D144
TYPE	BCITS	4	Specifies the data type of the column host variable. Valid values include: CHAR DATE DECIMAL FLOAT GRAPHIC INTEGER LONG VARCHAR LONG VARGRAPHIC SMALLINT TIMESTAMP TIME VARCHAR VARCHAR VARGRAPHIC	D141, D144

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DELETE Statement

The following list contains parameters that are unique to the DELETE statement. See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
FUNC	BCIT	8	Access code e.g., GN, GHU	S145, S185 S146, S149
GENKEYL	BC	30	<i>hvname</i> nl Generic key length	S146
OPCODE	BCIT	5	Ssa-opcode GTEQ EQUAL	S147, S145, S146
RECLTH	BC	60	<i>hvname</i> nl Record-length	S146
WHERE	BCITS	240	(DB2 only) Specifies the WHERE condition to be used in the generated USER I/O	S147

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DEVICE Statement

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
EATTR	I	1	Y N	S163

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			Y = extended attributes supported	
FEAT	I	60	(feat1] ,feat2]..[,featn]) Defines the associated MFS features	S163
TYPE	I	10	Device-type MFS-supported device ex. 3270-A4	S163

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DLIDSC Statement

This statement defines DL/I or EXEC DLI qualification options. For DL/I, each DLIDSC causes the generation of an SSA. For EXEC DLI, qualification parameters impact the generated data access (and possibly working storage).

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
BOOLEAN	BCIT	3	Specifies the Boolean operator(s) used to connect qualification statements in the case that multiple qualification statements have been defined.	D118
CMDCODE	BCIT	4	Specifies the default command code to be used in generated calls referencing this DLIDSC	D116, D118
CONCATK	BCIT	1	Y N Y = a concatenated key will be used to qualify an I/O request N = qualification statements will be an I/O request	D118
CURRENT	BCIT	1	Y N Y = position is to be maintained for all levels above the segment type CURRENT modifies for this I/O request	D118
DBDNAME	BCIT	8	<i>dbdname</i>	D100, D111, D112, D116, D118
DBMS	BC	8	Identifies the DBMS environment Valid values include DL/I EXEC DLI VSAM SEQ DB2	S165, B168
IMSKEY	I	8	<i>fldname(s)</i> defined in the segment (usually the key	D116, D118

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			field)	
IOAREA	BCIT	60	Specifies the I/O area to be used by USER I/O referencing this DLIDSC	D118
KEY	BCIT	60	Specifies the name of the data area(s) in the program containing the value(s) to be compared to the field(s) identified by IMSKEY	D116, D118
KEYFEED	BCIT	30	Specifies the key feedback area to contain the concatenated key after a successful call	D118
KEYLTH	BCIT	3	Key length	D111, D115
KEYPIC	BCIT	23	ssa key forward when other than character	D115, D118
label	BCIT	8	Name of DLIDSC to be referenced by USER I/O	D116, D118
LOCKED	BCIT	1	Y N Y = causes the segment to be reserved for exclusive use by the program; tells DL/I to keep a segment from being altered by other programs until the next sync point.	D118
NAME	BCIT	8	<i>segname</i>	D111, D112, D116, D118
OFFSET	BC	4	Offset to parent segment in the I/O area	D118
OPCODE	BCIT	2	Indicates how DL/I is to compare the two values specified in the qualification statement	D118
OPTION	BCIT	1	F L F = First command option L = last command option	D118
PARENTG	BCIT	1	Y N Y = override the parentage normally set by DL/I	D118
PATH	BCIT	1	Y N Y = segment referred to by this DEFQUAL will be retrieved as part of a path call	D118
PROCSEQ	BCIT	8	Secondary Index Name	D118
SEGLBL	BCIT	8	segment label	S125
SEGLTH	BCIT	60	<i>hvname</i> nl record-length	D111
<i>segname</i>	BCIT	8	<i>segname</i>	D111, S125

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
VARLTH	BCIT	1	Y N Y = segment is a variable-length segment	D118

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DLIPSB Statement

This statement requests the generation of PSB for screens that access DL/I files.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
PSBNAME	BC	8	<i>psbname</i> override	F113, S165, B168

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

DRIVER Statement

This statement indicates that the program is an online driver program in a statically-linked system. This statement also defines the online driver program's characteristics.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
APPLID	I	7	Application identifier	S210
CMPLOPT	I	253	(parameter1 [,parameter2,...]... [parametern])	S110
CRTDATE	I	8	YY/MM/DD	—.row
DESC	I	40	Driver Definition description	S100, S210
FRSTPGM	I	5	Screen ID of first screen to receive control	S210
HEADER	I	5	Header identifier	S100
HOLD	I	1	Y N Y = generated driver program will use Hold	S210
ID	I	5	Driver identifier	S100
IDENTIF	I	8	<i>mbrname</i>	S210

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
INIT	I	8	<i>mbrname</i>	S210
LANG	I	3	COB PL/I	S210
PGMCUST	I	240	(section1,mbrname1 [,section2,mbrname2]... [,sectionn,mbrnamen])	S210
PROCDR	I	8	<i>mbrname</i>	S210
REMARKS	I	8	<i>mbrname</i>	S210
SECTION	I	240	(mbrname1[,mbrname2]... [mbrnamen])	S210
TERM	I	8	<i>mbrname</i>	S210
UPDDATE	I	8	YY/MM/DD	-
UPDTA	I	1	Y N	S210
UPDTIME	I	4	HH:MM	-
UPDUSER	I	8	User ID that last updated the program in the TDF. -	
WKAREA	I	240	(mbrname1[,mbrname2]... [mbrnamen])	S210
XFER	I	8	<i>mbrname</i>	S210
XFERWKA	I	240	(mbrname1[,mbrname2] ... [,mbrnamen])	S210

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

Note: IMS Driver programs (DRs) *must* be exported/generated with ENV=I. They cannot be generated with ENV=T.

ERASE Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

FIELD Statement

This statement specifies the characteristics of each field on the screen, including the following:

- Field's usage (output, input, outin, or select)
- Field's position and length
- Screen attributes (i.e., protection, intensity, highlighting, and color)
- Source (on output) or target (on input) storage area
- Field edit routine to be executed for the field variable name
- Formatting, conversion, and acceptable values or ranges for input fields
- Name of the associated HELP text key, if any

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ATTRINT	CIT	6	NORMAL HIGH BLANK Display intensity	P158, P180, P181, P182
ATTRPRO	CIT	1	Y N Y = field is protected	P158, P181, P182
CALC	B	240	Arithmetic expression	
CNTGRP	B	9	<i>grpname</i> ALLDETAIL name of group to count	
CONVERT	BCIT	248	(screen-val1,stored-val1 [,screen-val2,stored-val2]... [,screen-valn,stored-valn])	P181, P182
DBNAME	BCIT	180	hvname1 ([out-hvname,] in-hvname1 [,in-hvname2])	P182 P155, P181
EACOLOR	CIT	2	BL RE PI TU YE NE GR DE blue, red, pink, turquoise, yellow, neutral, green, default	P158, P180, P181, P182 P181, P182
EAHIGH	CIT	2	BL UN RE DE blink, underline, reverse, default	P158, P180, P181, P182
EAVVALID	CIT	2	MF ME BO Must fill, must enter, both	P158, P181, P182
FLDTYPE	BCIT	7	Field edit (fldtype1, fldtype2)	P155, P181 P182
FMTCNTL	I	1	MFS	P181

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			MIDONLY MIDONLY,{MFS-keyword pfkey <i>fldname</i> }	
FMTEXTIT	I	6	(<i>exitnum</i> , <i>exitvect</i>) <i>exitnum</i> = the MFS exit routine number <i>exitvect</i> = a value to be passed to the exit routine three bytes per parameter	P181
FORMAT	BCIT	248	'format-mask ex. ' XXX-XXXX-XXX '	P181, P182
HELPMMSG	CIT	25	Message-name	P156, P181, P182
IEXTEND	CIT	472	(<i>hvname1</i> [, <i>hvname2</i>] ... [, <i>hvnamen</i>]) Edit parameters	P186
INDBIO	CIT	1	Y N Y = execute H-100-INPUT-TERM section automatically	P182
INEDIT	CIT	1	Y N Y = execute E-100-INPUT-EDITS section automatically	P182
INIT	BCIT	60	Initial value	P181, P182,
label	BCIT	8	Field name	P155, P180, P181, P182,
LTH	BCIT	3	Field length	P155, P156, P159, P158, P180, P181, P182
MAPOUT	BCIT	60	<i>hvname</i>	P181
NEXTPGM	CIT	5	Next program ID on SELECT field	P182
OEXTEND	BCIT	472	(<i>hvname1</i> [, <i>hvname2</i>] ... [, <i>hvnamen</i>])	P186
OF	BCIT	70	<i>hvname</i> ([out- <i>hvname</i> ,] in- <i>hvname1</i> [,in- <i>hvname2</i>])	P181, P182
OUTATTR	IT	1	Y N Y = output attributes will be included in the output buffer	P158, P180, P181
PIC	BCIT	28	' Picture ' '9(3) COMP'	P181

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
POS	BCIT	5	(ll,ccc) ll = line position ccc = column position	
RANGE	CIT	248	(start-range1,end-range... [,start-rangen,end-rangen])	P182
REQ	CIT	1	Y N C Y = field must be entered N = field is optional C = field must be entered under the conditions specified by consistency code	P155, P181
SCONSIS	CIT	8	<i>mbrname</i>	P182
SCOPE	B	6	<i>grpname</i> PAGE REPORT	
SELKEY	CIT	112	<i>hvname, hvname</i> Screen-key- <i>fldname</i> , stored-key- <i>fldname</i>	P182
TEXT	BCIT	30	LITERAL field text	P155, P180
TOTREF	B	8	<i>fldname</i>	
TOTSIZE	B	4	(ld,rd) Ld = number of left decimal positions Rd = number of right decimal positions	
usage	BCIT	8	INPUT OUTPUT OUTIN SELECT LITERAL	P180, P181, P182
VALUES	CIT	248	(value1 [,value2] ... [,valuen])	P182

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

GETDIAG Statement

The Generator uses the GETDIAG macro to specify GET DIAGNOSTICS statements for DB2 data access requests.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
CONDPTR	BCITS	32	Number or host variable containing a number	S240, S242, S243
DIAGS	BCITS	240	Comma-separated paired list of codes and corresponding host variables (code1,hostv1,code2,hostv2,...)	S240, S241, S242, S243
GDCLSC	BCITS	8	Custom code mbrname (only applicable to data access requests containing cursors). This custom code point is associated with the CLOSE CURSOR statement.	S240, S241, S242, S243
GDCUST	BCITS	8	Custom code mbrname. This custom code point is available to all GET DIAGNOSTICS specifications.	S240, S241, S242, S243
GDOPNC	BCITS	8	Custom code mbrname (only applicable to data access requests containing cursors). This custom code point is associated with the OPEN CURSOR statement.	S240, S241, S242, S243
LOCFLAG	BCITS	6	IMBED G100 COMMIT ROLLB IMBED = generates GET DIAGNOSTICS statement immediately following EXEC SQL statement G100 = generates GET DIAGNOSTICS statement in a G-100 (or G_100) section COMMIT = generates GET DIAGNOSTICS statement following the COMMIT statement ROLLB = generates GET DIAGNOSTICS statement following the ROLLBACK statement	S240, S241, S242, S243
NBR	BCITS	2	Number (1 through 8)	S240
SVCODE	BCITS	1	Y N Y = generates code to save and restore previous SQLCODE value N = does not generate code to save and restore previous SQLCODE value	S240, S241, S242, S243

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
TYPE	BCITS	4	STMT COND COMB CUST	S240

GROUP Statement

This statement identifies the beginning of a logical item or entry of one or more lines to be printed in a batch report.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
CTLLTH	B	3	Control variable length	
CTLPIC	B	30	'picture'	
CTLVAR	B	60	<i>hvname</i>	
FMTCUST	B	8	<i>mbrname</i>	
FORGRP	B	240	(grpname1[,grpname2] ... [,grpnamen]) ALLDETAIL	
label	B	8	Group name	
MINOR	B	8	<i>grpname</i> Minor CONTROL group	
PRINT	B	60	<i>hvname</i>	
REPSEQ	B	42	(n1[,n2] ... [nn]) Line skipping sequence	
SKIPAF	B	4	n PAGE Skip n lines after printing	
SKIPBEF	B	4	n PAGE Skip n lines before printing	
TDSKIP	B	2	n Skip n lines before printing first detail	
<i>Gptype</i>	B	7	TOPPAGE TOPDTL DETAIL BOTPAGE CONTROL SUMMARY Group type	

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

HOLD Statement

The following list contains parameters that are unique to the HOLD statement.
See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
HOLDLTH	IT	6	Specifies the length required to HOLD the transfer area information	Not on TDF

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

IMSDRV Statement

This statement requests a generation of an online IMS driver program and defines its characteristics.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
A4MSG	I	40	Literal <i>hvname</i> Value to be displayed in ERRMSG1 when A4 error status is received after IMS message switch	4.21
A4PGM	I	1	Screen ID Program to receive control when A4 error status is received after IMS message switch	4.21
CONVERS	I	1	Y N Y (conversational) = The system will use an IMS SPA N (non-conversational) = The system will use a WORKSPA database	F113, 4.21
FRSTMOD	I	8	MFS MOD name for the first screen	4.21
GENPCBS	I	1	Y N Y = generate PCB masks N = include masks coded in LNKCOPY and USGCOPY members	F113
IOASIZE	I	5	Largest segment area size	
LINKDYN	I	1	Y N Y = dynamically link to any static subroutines not specified in the LINKPGM, MSGPGM, MSGTBL, or	

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
MSGTRAN parameters				
LINKPGM	I	180	(id1[,id2] ... [,idn]) ANY Screen IDs where control is transferred by calling statically linked subroutines	4.21
LNKCOPY	I	8	<i>mbrname</i>	F113
MSGBUF	I	8	(<i>mbrname</i> , length) User-defined buffer area for automatic message switching	4.21
MSGPGM	I	180	(id[,id2] ... [,idn]) ANY Screen ID for automatic message switching when transaction code is equal to the program name	4.21
MSGTBL	I	11	(table- <i>mbrname</i> , number-of-entries)	4.21
MSGTRAN	I	180	(id1,tran1...[,idn,transn]) Screen ID and corresponding IMS transaction code for automatic message switching	4.21
PGMNAME	I	8	Program name override	4.21
SPASIZE	I	5	IMS spase in bytes	4.21
TPISIZE	I	5	Maximum input buffer size	4.21
TPOSIZE	I	5	Maximum output buffer size	4.21
TRACE	I	1	Y N Y = generate code to maintain CA Telon trace variables	4.21
TRANCDE	I	10	IMS transaction code override	4.21
USGCOPY	I	8	<i>mbrname</i>	
WKSPAIN	I	1	Y N Y = generate CA Telon WORKSPA database initialization code in program section C-920-GET-WORKSPA N = do not generate CA Telon WORKSPA database initialization	4.21
WKSPAIO	I	16	(<i>mbrname</i> , <i>mbrname</i>) <i>Getwkspa</i> and <i>putwkspa</i>	4.21
WKSPASZ	I	5	Size of the WORKSPA database	4.21

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

IMSMFS Statement

This statement requests a generation of the IMS MFS control blocks for a screen.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LINEOPT	I	1	1 2 3 1 = Use CA Telon line optimizing subroutine 2 = Simulate subroutine optimization in procedural custom code 3 = Do not generate line optimizing code	S162
MFSMOD	I	8	MFS MOD name override	S162, S163
SEGEXIT	I	6	(<i>exitnum</i> , <i>exitvect</i>) <i>exitnum</i> = the MFS exit routine number <i>exitvect</i> = a value to be passed to the exit routine	S163
SYMSG	I	8	<i>fldname</i> Destination of system messages	S162, S163
TRANCDE	I	10	MID transaction code	S162
TRANFLD	I	8	<i>Fldname</i> containing transaction code	S162
TRANMFS	I	1	Y N Y = MFS processes the transaction code	S162

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

IMSPGM Statement

This statement requests a generation of an IMS program and defines its characteristics.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
A4EMSG	I	40	literal <i>hvname</i> Value to be displayed in ERRMSG1 when A4 error status is received after IMS message switch	S162
A4EPGM	I	6	Screen ID	S162

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			Program to receive control when A4 error status is received after IMS message switch	
CONVERS	I	1	Y N Y = the program uses IMS conversational processing (SPA)	S162
GENPCBS	I	1	Y N Y = generate PCB masks N = include masks coded in LNKCOPY and USGCOPY members	F113, S162
LINEOPT	I	1	1 2 3 1 = Use CA Telon line optimizing subroutine 2 = Simulate subroutine optimization in procedural custom code 3 = Do not generate line optimizing code	S162
LINKOPT	I	1	D S D = dynamic link S = static link	F113, S162
LINKPGM	I	180	(id[,id2] ... [,idn]) ANY	S162
LNKCOPY	I	8	<i>mbrname</i>	F113, S162
MFSMOD	I	8	MFS MOD name for this screen	S162
MSGBUF	I	8	(mbrname, length) User-defined buffer area for automatic message switching	S162
MSGPGM	I	180	(id1[,id2] ... [idn]) ANY	S162
MSGTBL	I	11	(table-mbrname, number-of-entries) Note: <i>mbrname</i> is 8 bytes, number of entries is 3 bytes	S162
MSGTRAN	I	180	(id1,tran1...[idn,trann])	S162
PGMNAME	I	8	Program name override	S162
SPACMPT	I	1	Y N Y = generate a compatible SPA for use by both static and dynamic program N = generate a SPA with a different number of overhead bytes based on whether the module is static or dynamic	S162
SPASIZE	I	5	Size of the SPA	S162

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
TRACE	I	1	Y N Y = generate trace variables	S162
TRANCDE	I	10	IMS transaction code override	S162
TRANFLD	I	8	<i>fldname</i> containing the transaction code	S162
USGCOPY	I	8	<i>mbrname</i>	B168
WKSPAIN	I	1	Y N Y = generate CA Telon WORKSPA database initialization code in program section C-920-GET-WORKSPA N = do not generate CA Telon WORKSPA database initialization	S162
WKSPAIO	I	16	(<i>mbrname</i> , <i>mbrname</i>) GET WORKSPA member PUT WORKSPA member	S162
WKSPASZ	I	5	Concatenated size of the WORKSPA data base segments	S162

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

IMSPSB Statement

This statement requests a generation of an IMS PSB.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
PSBNAME	I	8	<i>psbname</i> override	S162

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

INREAD Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

JOINCOL Statement

This statement defines the column names (and the corresponding table) to be used to generate the JOIN criterion.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
JREF1	BCITS	240	Specifies the first column name (and the table the column is from) to be used in the generated join where criterion.	Not on TDF
JREF2	BCITS	240	Specifies the second column name (and the table the column is from) to be used in the generated join where criterion.	Not on TDF

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

JOURNAL Statement

The parameters for this statement are as follows:

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
FUNC	C	4	Valid values: WAIT or spaces (i.e., issue " EXEC CICS WAIT JOURNAL " call or " EXEC CICS JOURNAL " call) Generic key length	S14J
IOAREA	C	60	Override queue definition copy/include member name	S14J
<i>label</i>	C	8	U-100 extension name (e.g., U-100-label-segname)	S14J
LTHLBL	C	30	Overrides default LTHOPT variable name	S14J
LTHOPT	C	1	Generate LTHOPT option (N Y)	S14J
NOSUSP	C	1	Generate NOSUSPEND option (N Y)	S14J
PFXLBL	C	30	Overrides default PFXLTH variable name	S14J

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
PFXLTH	C	1	Generate PFXLTH option (N Y)	S14J
REQID	C	1	Generate REQID option (N Y)	S14J
REQIDLBL	C	30	Overrides default REQIDLBL variable name	S14J
<i>segname</i>	C	8	<i>segname</i>	S14J
STARTIO	C	1	Generate STARTIO option (N Y)	S14J
WAIT	C	1	Generate WAIT option (N Y)	S14J

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

MATCH Statement

Export will create a MATCH statement that the Generator will use to produce a MATCH program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ENDTRAN	B	B	custom code member name	B110
INMAST	B	8	custom code member name	B110
INTRAN	B	8	custom code member name	B110
MATCH	B	8	custom code member name	B110
MGREATR	B	8	custom code member name	B110
TGREATR	B	8	custom code member name	B110

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

MATCHKEY Statement

The Generator uses the MATCHKEY statement to produce matching logic for each match key in a MATCH program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
FORM	B	1	Form of key - A = Alphabetic, N = Numeric	B1MA
KEYNUM	B	2	Hierarchical position of key field	B1MA
LTH	B	4	Length of key	B1MA
MASTKEY	B	60	Name of master file key field	B1MA
SIZE1	B	2	Digits left of decimal	B1MA
SIZE2	B	2	Digits right of decimal	B1MA
TRANKEY	B	60	Name of transaction file key field	B1MA

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

MATCHX Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

MERGE Statement

The Export utility produces the MERGE statement as an intermediate step to Generating a MERGE program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DATA SET	B	160	From 1 to 20 merge file names, of eight bytes each	B1M2

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

MERGE# Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

MERGEGRP Statement

The Generator uses the MERGEGRP statement to produce MERGE logic for each key group in a MERGE program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DESC	B	40	Description of the merge key group	B1M1
FORM	B	1	Form of a merge key group table field: A=alphanumeric or N=numeric .	B1M1
GROUPNUM	B	2	Hierarchical position of key group	B1M1
LTH	B	4	Length of a merge key group field (to create a save area table)	B1M1
SIZE1	B	2	Picture clause: number of digits before the decimal point	B1M1
SIZE2	B	2	Picture clause: number of digits after the decimal point	B1M1

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

MERGEKEY Statement

The Generator uses the MERGEKEY statement to produce MERGE logic for each key in a MERGE program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DATA SET	B	8	Name of data set with which key is associated	B1M2
DATANAME	B	60	Name of DATA SET's merge key field	B1M2

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

NONTERM Statement

The following parameters can be specified for the NONTERM CA Telon statement:

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
APPLID	C	7	Application identifier used to construct program names at generation.	N110
CMPLOPT	C	253	(parameter1 [,parameter2,... [parametern]])	N110
CRTDATE	C	8	YY/MM/DD	— .row
DESC	C	40	Description for display only.	N110
GETTRAN	C	8	Name of custom code member to be placed in the C-N100-GET-TRAN section of the generated program after AUTOEXEC Data Access calls.	N110
HEADER	C	2	Unique program identifier within header.	N110
ID	C	4	Unique program identifier within HEADER (see below); positional field (appears immediately after NONTERM).	N110
IDENTIF	C	8	<i>mbrname</i>	N110
INIT1	C	8	Name of custom code member to be placed BEFORE code that validates printer ID (if any) in Q-N100-PROGRAM-INIT.	N110
INIT2	C	8	Name of custom code member to be placed AFTER code that validates printer ID (if any) in Q-N100-PROGRAM-INIT.	N110
LANG	C	3	Programming language in which code is to be generated (COB=COBOL, PLI=PL/I).	N110
LANGLVL	C	3	CA Telon release level identifier.	N110
PGMCUST	C	240	Pair(s) of section and custom code member names: the latter are to be copied into the former.	N110
PRCTRAN	C	8	Name of custom code member to be placed in the A-N100-PROCESS-TRAN program after the CONTROL INDICATOR is set.	N110
PROCEDR	C	8	<i>mbrname</i>	N110
PRTDEST	C	4	Printer ID to which report is to be routed.	N110
REMARKS	C	8	Custom code member name for remarks at head of generated program.	N110

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
RPTDEST	C	8	Report output destination (report/name of REPORT data type defined in DG/Printer).	N110
SECTION	C	240	Name(s) of custom code members to be copied into the Z-N100-SECTIONS portion of the generated program, to be performed/called from other parts of the program.	N110
SIZE	C	6	Report size, in format 11,ccc where "11" is number of lines and "ccc" is number of columns.	N110
TERM	C	8	Name of custom code member to be placed in the T-N100-PROGRAM-TERM section of the generated program.	N110
UPDDATE	C	8	YY/MM/DD	
UPDTIME	C	4	HH:MM	
UPDUSER	C	8	User ID that last updated the program in the TDF.	
WKAREA	C	240	Names of copy/include member(s) containing work area definitions, to be copied into WORKING-STORAGE section (COBOL) or declarations area (PL/I) of generated program.	N110
XFERWKA	C	8	<i>mbrname</i>	N110

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

OUTREAD Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common Data Access parameters.

PANEL Statement

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
CRTDATE	BCIT	8	YY/MM/DD	
INPCHAR	CIT	1	INPUT field paint character default = <	F114
LITBRK	BCIT	1	Literal break field paint character default = \	F114

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
OICHAR	CIT	1	OUTIN field paint character default = +	F114
OUTCHAR	BCIT	1	OUTPUT field paint character default = >	F114
SELCHAR	CIT	1	SELECT field paint character default =	F114
SIZE	BCIT	5	(ll, ccc)	F114 row.
UPDDATE	BCIT	8	YY/MM/DD	-
UPDTIME	BCIT	4	HH:MM	-
UPDUSER	BCIT	8	User ID that last updated the program in the TDF.	
UPRLWR	BCIT	3	ON OFF ON = convert to upper case OFF = leave as lower case On the TDF screen P100, U is entered as ON, L as OFF	F114, P100

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

PARAM Statement

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DBNAME	S	60	Mapping to working storage variable	B2P1
DEC	S	3	Numeric – parameter scale (if applicable)	B2P1
IND	S	1	Y N Y = Generate indicator parameter	B2P1
KIND	S	1	I—IN, O—OUT, B—INOUT	B2P1
LTH	S	5	Numeric—Parameter length	B2P1
NAME	S	18	Parameter name	
TYPE	S	1	Parameter type (e.g., C—CHAR, I—INTEGER, V—VARCHAR...)	

PLIXOPT Statement

For applications written in PL/I, this statement overrides specific PL/I defaults defined for the application.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ALIGN	BCIT	1	A U A = Aligned U = Unaligned	S164
<i>env</i>	BCIT	5	CICS IMS TSO BATCH ALL	S164
REORDER	BCIT	1	Y N Y = use PL/I compiler REORDER option	S164
STORAGE	BCIT	1	S A S = Static A = Automatic	S164
XOPTS	BCIT	60	'execution options'	S164

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

READ Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common Data Access parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LTHLBL	C	30	Overrides default generated COBOL or PL/I queue LTHOPT variable or record-level LTHLBL	S14Q

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

READNEXT Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common Data Access parameters.

RECORD Statement

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
AUXMAIN	C	1	A M (AUX/MAIN)- Type of storage to be used for temp storage (TS)	D11Q, S13Q
COBDIV	B	2	FD WS	D114, D114
COBVSky	B	60	(hvname[,DUPLICATE])	D114, D114
COPY	BCIT	8	Copy/include member name override	D114, D114
COPYLBL	BCIT	30	Data area name <i>hvname</i> I/O area name	D114, D114
COPYLV1	BCIT	1	Copy member is level 01 (N Y) Y = I/O area mbrname starts at 01 level N = I/O area mbrname starts at level > 02	D114, D114
DBSEG	BCIT	8	segname	S125
GENKEYL	C	30	<i>hvname</i> nl	D114, D114
KEY	BCIT	60	<i>hvname</i> Key field	D114, D114
KEYLEN	BCIT	3	Key length	D114, D114
LABEL	BCIT	8	Default Label to be associated with the dateset, queue or journal	D114, D11J, D11Q, S125
LTHLBL	BC	30	Overrides default LTHOPT variable name (Valid for QUEUES and JOURNALS only)	S13Q
LTHOPT	BC	1	Indicates whether or not LTHOPT option should be used in I/O requests; (N Y) (Valid for QUEUES and JOURNALS only)	S13Q
<i>name</i>	BCIT	8	DATASET, CQUEUE, or CJOURNAL name	D114, S136, S13J, S13Q
OPCODE	BCIT	2	ssa-opcode GETQ EQUAL	D114, D114
OPTLIST	C	60	(option1[,option2] ... [,optionn]) Values = RRN, SEGSET, SEGETALL, SYSID, MASSINSERT, DEBKEY, DEBREC	D114, D114
PFXLBL	C	30	Name of the COBOL or PL/I containing the prefix	S13J

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
(CICS journal only)				
PFXLTH	C	4	Length of user prefix data (CICS journal only)	S13J
QUELBL	C	30	Overrides CICSQUE name variable in the "EXEC CICS" commands (Valid for QUEUES only)	S13Q
RECLTH	BC	60	<i>hvrname</i> nl Record-length	D114, D114
SYSID	C	4	System ID to be used when accessing a queue (Valid for QUEUES only)	
Usage	BCIT	7	BROWSE DEFINE @DEFINE @DUMMY DUMMY	S125, D114

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

REPLACE Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common Data Access parameters.

REPORT Statement

This statement indicates that any output should be routed to an IMS/DC printer (rather than to a terminal screen). This statement also defines characteristics of the print subroutine that is built for this type of definition.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
APPLID	I	7	Application identifier	S310
CMPLOPT	I	253	(parameter1 [,parameter2,... [parametern]])	S310
CRTDATE	I	8	YY/MM/DD	-
DESC	I	40	Report Definition description	S100, S310
HEADER	I	5	Report identifier	S100
ID	I	5	Report identifier	S100
IDENTIF	I	8	<i>mbrname</i>	S310
LANG	I	3	COB PLI	S310

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LINKWKA	I	8	<i>mbrname</i>	S310
OINIT1	I	8	<i>mbrname</i>	S310
OINIT2	I	8	<i>mbrname</i>	S310
OUTTERM	I	8	<i>mbrname</i>	S310
PGMCUST	I	240	(section1,mbrname1 [,section2,mbrname2] ... [,sectionn,mbrnamen])	S310
PROCEDR	I	8	<i>mbrname</i>	S310
REMARKS	I	8	<i>mbrname</i>	S310
SECTION	I	240	(mbrname1[,mbrname2]... [,mbrnamen])	
SIZE	I	5	(ll,ccc) lines and columns	S310
UPDDATE	I	8	YY/MM/DD	
UPDTIME	I	4	HH:MM	
UPDUSER	I	8	User ID that last updated the program in the TDF.	
WKAREA	I	240	mbrname1 (mbrname1[,mbrname2] ... [,mbrnamen])	S310
XFERWKA	I	240	(mbrname1[,mbrname2[... [,mbrnamen])	S310

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

ROW Statement

This statement identifies the SQL table or TLNJOIN to which the user exec/I/O pertains, which follows the ROW statement.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
KEY	BCITS		Specifies the default hvname(s) to be used in the generated USER I/O WHERE condition.	S125, S137
label	BCITS	8	Specifies the CA Telon identifier for the ROW to be used in the generation of USER I/O procedure or paragraph names.	S125
TBLNAME	BCITS	18	Table name	S137

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
TBLQUAL	BCITS	8	Qualifier for table name	S137
TMPCOMT	BCITS	6	Temporary table commit options: SAVE DELETE DROP	S137
TMPNAME	BCITS	18	Temporary table name	S137
TMPQUAL	BCITS	8	Qualifier for temporary table	S137
usage	BCITS	8	DEFINE DUMMY @DEFINE @DUMMY HOLD HOLD WORKSPA @HOLD @WORKSPA BROWSE	S125, S137
WHERE	BCITS	240	Specifies the default WHERE condition to be used in the generated USER I/O.	S137

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SCREEN Statement

This statement indicates that the program processes a screen. It also specifies screen characteristics, including:

- Application and screen identifying information
- Control data
- Program remarks
- Cursor positioning
- The program to which control passes from this screen
- Branching logic (based on PF-key processing)
- The name of custom code blocks within parameters
- Work areas and transfer work areas via custom code block names
- The indication of whether to use the system HELP or HOLD facilities

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ALARM	CIT	1	Y N Y = ring terminals alarm on ERROR-ATTR	F112, S112
APPLID	CIT	7	Application identifier	S210
CAPS	CIT	1	ON OFF ON = translate lower case to upper on input	F112, S112

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
OFF = do not translate characters on input				
CMPLOPT	IC	253	(parameter1 [,parameter2,... [parametern]])	S110
CON SIS	CIT	8	<i>mbrname</i>	P100, S110
CRTDATE	CIT	8	YY/MM/DD	-
DESC	C	40	Description for display only.	N110
CURSCUS	CIT	8	<i>mbrname</i>	S110
CURSOR	CIT	8	<i>fdname</i> Cursor position	S110
DESC	CIT	40	Screen Definition description	S100, S110
EAERR	CIT	7,5	BLUE RED PINK GREEN TURQ BLINK YELLOW REVER NEUTRAL UNDER DEFAULT	F112, S112
EAIN	CIT	7,5	BLUE RED PINK GREEN TURQ BLINK YELLOW REVER NEUTRAL UNDER DEFAULT	F112, S112
EALIT	CIT	7,5	BLUE RED PINK GREEN	F112, S112

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
			TURQ BLINK YELLOW REVER NEUTRAL UNDER DEFAULT	
EAOUT	CIT	7,5	BLUE RED PINK GREEN TURQ BLINK YELLOW REVER NEUTRAL UNDER DEFAULT	F112, S112
EATTR	CIT	1	Y N Y = use extended screen attributes	F112, S112
EOFKEY	IT	1	Y N Y = EOF key is used to delete fields N = EOF key cannot be used to delete fields	F112, S112
FLDEDIT	CIT	8	<i>mbrname</i>	S110
HEADER	CIT	5	Header identifier	S100
HELP	CIT	1	Y N Y = HELP will be used	F112, S112
HOLD	CIT	1	Y N Y = HOLD will be used	F112, S112
ID	CIT	5	Screen identifier	S100
IDENTIF	IC	8	<i>mbrname</i>	S110
ININIT1	CIT	8	<i>mbrname</i>	S110
ININIT2	CIT	8	<i>mbrname</i>	S110
INTERM	CIT	8	<i>mbrname</i>	S110

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LANG	CIT	3	COB PL/I	S110, S210
NEXTPGM	CIT	5	Next program ID	S110
OINIT1	CIT	8	<i>mbrname</i>	S110
OINIT2	CIT	8	<i>mbrname</i>	S110
OUTIFIL	CIT	1	B U N Output fill character B = spaces U = underscores N = low-values	F112, S112
OUTTERM	CIT	8	<i>mbrname</i>	S110
PFKEYS	CIT	240	(name1[,name2] ... [,namem]) name is a substring of a PF key <i>mbrname</i> . The size of each name is installation dependent.	S110
PGMCUST	CIT	240	(section1. <i>mbrname</i> 1 [,section2. <i>mbrname</i> 2]... [,sectionn. <i>mbrname</i> n])	S110
PROCEDR	IC	8	<i>mbrname</i>	S110
REFRESH	CIT	1	Y N Y = save output buffer in the screen image area	
REMARKS	CIT	8	<i>mbrname</i>	S110
SECTION	CIT	240	(<i>mbrname</i> 1[, <i>mbrname</i> 2] ... [, <i>mbrname</i> n])	S110
SIZE	CIT	6	(ll,ccc)	S110
UPDDATE	CIT	8	YY/MM/DD	
UPDTA	CIT	1	Y N create update area in working storage	
UPDTIME	CIT	4	HH:MM	
UPDUSER	CIT	8	User ID that last updated the program in the TDF.	
WKAREA	CIT	240	(<i>mbrname</i> 1[, <i>mbrname</i> 2] ... [, <i>mbrname</i> n])	S110
XFERWKA	CIT	240	(<i>mbrname</i> 1[, <i>mbrname</i> 2] ... [, <i>mbrname</i> n])	F112, S110

* B = Batch; C = CICS; I = IMS; T = TSO

SEGEDIT Statement

This statement requests that, at program generation time, CA Telon create code to access a database segment/record to verify its existence and handle resulting errors.

Note: For all fields, commas or spaces can be used as separators.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
CMDCODE	CIT	4	SSA command code	P168
CURSOR	CIT	8	<i>fldname</i>	P168
DESC	CIT	64	SEGEDIT description	P161
ERRCHAR	CIT	60	(<i>fldname1</i> [, <i>fldname2</i>] ... [, <i>fldnamen</i>])	P168
ERRMSG	CIT	120	<i>hvname</i> ' literal ' Error message	P168
ERROR	CIT	8	FOUND NOTFOUND UNIQUE NOTUNIQUE DUPLICATE NOTDUPLICATE	P168
FUNC	CIT	4	Access code e.g., GN, GHU	P168
GENKEYL	C	60	<i>hvname</i> nl Generic key length	P168
HILIGHT	CIT	60	(<i>fldname1</i> [, <i>fldname2</i>] ... [, <i>fldnamen</i>])	P168
IOAREA	CIT	60	<i>hvname</i> Data area name	P168
KEY	CIT	60	<i>hvname</i> Key field	P168
label	CIT	8	Edit name from Panel Specification Menu	
OPCODE	CIT	5	ssa-opcode GTEQ EQUAL	P168
PCBNAME	CIT	12	<i>pcbname</i>	P168
QUALIFY	CIT	1	Y N Y = use the alternate SSA defined for the segment on the SEGMENT statement N = use the standard SSA defined for the segment on the SEGMENT statement.	P168

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
SEGLTH	C	60	<i>hvname</i> nl Record length	P168
SEGLOOP	CIT	1	Y/N Indicates whether or not SEGEDIT should be performed in a loop for a segloop field.	P168
<i>segname</i>	CIT	8	Segment name	P168
SSALIST	CIT	180	(<i>hvname1</i> [, <i>hvname2</i>] ... [<i>hvnamen</i>]) <i>ssa1</i> , <i>ssa2</i>	P168
WHEN	CIT	180	(<i>cond1</i> [, <i>connector1</i> , <i>cond2</i>] ... [, <i>connectorn-1</i> , <i>condn</i>]) Cond is <i>hvname</i> or an operator combination. Operators are EQ, GE, GT, LT, LE, and NE. Connectors are AND, OR and THENIF.	
WHERE	CIT	240	Where field equals value. Field equals column on DB2 table. Where clause is DB2 filter criteria.	P168

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SEGEN Statement

This statement identifies the end of loop processing. It has no parameters.

SEGLOOP Statement

This statement identifies the beginning of loop processing and defines the characteristics of the loop, including:

- Whether the loop processes input, output, or outin fields
- Line spacing (down the page)
- Column spacing (across the page)
- The use of paging
- Custom code block names
- Database search-key information
- Information necessary to maintain an array of records included in the loop (key values, array index, first-key value, and so on)

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
CINCRE	CIT	28	(col-spacing1[,col-spacing2]... [,col-spacingn]) Number of columns to be skipped between loop iterations	P170, P175
COLSGLP	CIT	1	Column SEGLOOP flag. Values are: Y N. Y = SEGLOOP loaded in column/row order N = SEGLOOP loaded in row/column order (default)	P170, P175
ICTLNM	CIT	8	<i>fldname</i> Field that, when it is filled with spaces, halts input processing	P170, P175
ICUST1	CIT	8	<i>mbrname</i>	P170, P175
ICUST2	CIT	8	<i>mbrname</i> NOTE: Replaces the ICUSTOM parameter	P170, P175
INCRE	CIT	28	(line-spacing1[,line-spacing2]... [,line-spacingn]) Number of lines to be skipped between loop iterations	P170, P175
ISEGIDX	CIT	30	<i>hvname</i> Input loop index name	P170, P175
LINECNT	CIT	6	<i>fldname</i> Line number of the loop iteration	P170

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
OCUST1	CIT	8	<i>mbrname</i>	P170, P175
OCUST2	CIT	8	<i>mbrname</i>	P170, P175
OCUST3	CIT	8	<i>mbrname</i>	P170, P175
OSEGIDX	CIT	30	<i>hvname</i> Output loop index name	P170, P175
PAGE	CIT	1	Y N Y = use paging	P175
PAGEKEY	CIT	60	<i>hvname</i> Key-field used for paging	P175
PAGESAV	CIT	2	The number of screens the program can scroll backwards through	P175
PKYLTH	CIT	3	Length of paging key (in bytes)	P175
PKYUNIQ	CIT	1	Y N Y = key must be unique N = non-unique keys allows	P175
SAVEKEY	CIT	60	(<i>hvname1</i> , <i>hvname2</i> ... [, <i>hvnamen-1</i> , <i>hvnamen</i>]) Pairs of key-field and table-field variables	P170
SCHFLDC	CIT	30	<i>hvname</i> Data item containing search criteria	P175
SCHFLDI	CIT	8	Name of DL/I search field	P175
SCHFLDL	CIT	3	Length of SCHFLDC	P175
STBRKEY	CIT	30	<i>hvname</i> Data item containing key data to start the browse	P175
TYPE	CIT	8	TABLE FILE	P170, P175
<i>usage</i>	CIT	8	OUTPUT INPUT OUTIN	P175

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SEGMENT Statement

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
CMDCODE	BCIT	4	SSA command code	
COPY	BCIT	8	<i>mbrname</i> NONE	S135
COPYLBL	BCIT	30	<i>hvname</i> I/O area name	S135
COPYLV1	BCIT	1	Y N Y = I/O area <i>mbrname</i> starts at 01 level N = I/O area <i>mbrname</i> starts at level greater than 02	S135
DBSEG	BCIT	8	segname	
DSCREF	BCIT	240	Specifies the DLIDSC statements to be used in building the qualification statements or SSAs for this USER I/O request	S135
IMSKEY	BCIT	8	DL/I key name	S135
INDICES	BCIT	60	(index1[,index2] ... [indexn]) DL/I index-name Indices are DLI DBD names	S135
KEY	BCIT	60	<i>hvname</i> Key field value	S135
KEYLEN	BCIT	3	Key length	S135
KEYPIC	BCIT	60	ssa key forward when other than character	S135
<i>label</i>	BCIT	8	segment name	S125, S135
OPCODE	BCIT	2	ssa-opcode GTEQ EQUAL	S135
PARENT	BCIT	8	Parent segment name	D111
PROCOPT	BCIT	5	Value to be included in the corresponding SENSEG statement of PSB	S135
<i>usage</i>	BCIT	7	BROWSE WORKSPA DEFINE HOLD @HOLD @WORKSPA @DEFINE DUMMY @DUMMY	S125, S135

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SORT Statement

The Export utility produces and the Generator processes one SORT statement for each sort in a program (either mainline or user-defined) and one SORTKEY statement for each sort key defined to the sort whose CA Telon statement it follows.

The data for mainline and user-defined sorts is stored on two different database segments with identical structures.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
COLLATE	B	8	Collating sequence: E=EBCDIC ; A=ASCII; custom code member name = other	BIS2
COPY	B	8	Sort file layout copy member	BIS2
COPYLBL	B	30	Override for file's I/O area name (only used with COPYLV1)	BIS2
COPYLV1	B	1	The file layout begins with an "01" level (Y N). (Only used with COPYLBL.)	BIS2
DESC	B	40	Description of the sort; 'MAINLINE' for mainline sort	B1S1
FILEIN	B	8	(COBOL only) Name of sort input file (mutually exclusive with INPROC)	BIS2
FILEOUT	B	8	(COBOL only) Name of sort output file (mutually exclusive with OUTPROC)	BIS2
<i>label</i>	B	8	Sortname ; 'MAINLINE' for mainline sort	BIS2
LRECL1	B	4	Sort record's minimum logical record length	BIS2
LRECL2	B	4	Sort record's maximum logical record length	BIS2
PREFIX	B	4	(PL/I only) Sort prefix	BIS2
PROCIN	B	8	Name of custom code member containing sort input processing procedure (mutually exclusive with INFILE)	BIS2

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
PROCOUT	B	8	Name of custom code member containing sort output processing procedure (mutually exclusive with OUTFILE)	BIS2
STORAGE	B	4	(PL/I only) Sort main storage minimum allowed is 88K)	BIS2

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SORTKEY Statement

The SORTKEY statement is generated for each sort key specified for a mainline or user-defined sort.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DATANM	B	60	Dataname of field to be sorted (COBOL only)	
FORM	B	2	Form of sort field: CH=character , ZD = picture (zoned decimal), PD = packed decimal, BI = binary, FI = fixed point, FL = floating point	
LTH	B	3	Length of sort field	
SIZE1	B	2	Picture clause: number of digits before the decimal (COBOL only)	
SIZE2	B	2	Picture clause: number of digits after the decimal (COBOL only)	
SRTORDER	B	1	Sort order: A=ascending , D=descending	
START	B	4	Starting position of sort field	

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SPBROWSE Statement

The following list contains parameters that are unique to the SPBROWSE statement. See BROWSE statement for a list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
RESLTCC	CIT	8	Custom code point for result set cursor allocation	S147
RESLTNR	CIT	2	Numeric—number of result sets in stored procedure being called	S147
RESLTPR	CIT	8	Name of stored procedure producing result sets	S147

SPPARM Statement

This statement requests the generation of calling parameters for a stored procedure being called from another CA_Telon program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DEC	BCIT	3	Numeric—parameter scale (if applicable)	B2P1
IND	BCIT	1	Y N Y = Generate indicator parameter	B2P1
KIND	BCIT	1	I-IN, O-OUT, B-INOUT	B2P1
LTH	BCIT	5	Numeric—parameter length	B2P1
NAME	BCIT	18	Parameter name	
TYPE	BCIT	1	Parameter type (e.g., C-char, I-integer, V-varchar...)	

* B = Batch: C = CICS: I = IMS: T = TSO

SPPGM Statement

This statement requests generation of a CA_Telon stored procedure program, and defines its characteristics.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ASUTIME	S	5	Numeric—maximum time that a procedure can run	B268
COLLID	S	18	Execution collection package name	B268
COMRETN	S	1	Y N Y = COMMIT on return	B268
DBINFO	S	1	Y N Y = DBINFO flag is set	B268
DETERM	S	1	Y N Y = Procedure is DETERMINISTIC	B268
EXTSCUR	S	1	2-DB2, U-User, D-Definer	B268
FENCED	S	1	Y N Y = Procedure is fenced	B268
NULCALL	S	1	Y N Y = nulls are allowed	B268
PRMSTYL	S	1	N-General with nulls, G-General, D-DB2SQL, J-Java	B268
PROGTYP	S	1	M-main, S-subroutine	B268
RESULTS	S	2	Numeric – number of result sets returned	B268
SCHEMA	S	8	Schema name	B268
SQLMOD	S	1	M-Modifies SQL Data, N-No SQL, S-Contains SQL, R-Reads SQL Data	B268
STAYRES	S	1	Y N Y = stored procedure stays resident	B268
WLMENV	S	1	Workload manager environment name	B268

SPRDNEXT Statement

The following list contains parameters that are unique to the SPRDNEXT statement. See READNEXT for a complete list of common parameters. Procedure program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
RESLTCC	BCIT	8	Custom code point for result set cursor allocation	S147
RESLTNR	BCIT	2	Numeric—number of result sets in stored procedure being called	S147
RESLTPR	BCIT	8	Name of stored procedure producing result sets	S147

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SPTRNACT Statement

The following list contains parameters that are unique to the SPTRNACT statement. See TRANSACT for a complete list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
RESLTCC	B	8	Custom code point for result set cursor allocation	S147
RESLTNR	B	2	Numeric – number of result sets in stored procedure being called	S147
RESLTPR	B	8	Name of stored procedure producing result sets	S147

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

SRC Statement

This statement inserts a line of COBOL or PL/I custom code in Consistency Edit statements SEGEDIT and XFEDIT.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
<i>value</i>	CIT	65	Program statement text	

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

STORED Statement

This statement specifies the characteristics of a CA_Telon Stored Procedure program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
APPLID	S	7	Application Identifier	B210
CMPLOPT	S	253	(parameter1 [,parameter2,] [parameter _n])	B210
COBFCPY	S	16	(mbrname1.mbrname2) Each <i>mbrname</i> 8 bytes max.	B210
CRTDATE	S	8	YY/MM/DD	-
DESC	S	40	Stored definition description	B100, B210
HEADER	S	5	Program header	B100
ID	S	5	Program identifier	B100
IDENTIF	S	8	<i>mbrname</i>	B210
INIT1	S	8	<i>mbrname</i>	B210
INIT2	S	8	<i>mbrname</i>	B210
LANG	S	3	COB PL/I	B210
PGMCUST	S	240	(section1,mbrname1 [,section2,mbrname2] ... [section,mbrnamen]) (max of 50)	B210
PROCDR	S	8	<i>mbrname</i>	B210
REMARKS	S	8	<i>mbrname</i>	B210
RUNOPTS	S	8	(option1[,option2] ... [,option _n])	B210
SECTION	S	240	(mbrname1[,mbrname2] ... [,mbrnamen]) (max of 35)	B210
SPINIT	S	8	<i>mbrname</i>	B210
SPNAME	S	8	<i>mbrname</i>	B210
SPPROC	S	8	<i>mbrname</i>	B210
SPTERM	S	8	<i>mbrname</i>	B210
TERM	S	8	<i>mbrname</i>	B210
UPDDATE	S	8	YY/MM/DD	-
UPDTIME	S	4	HH:MM	-

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
UPDUSER	S	8	User ID that last updated the program in the TDF. -	
WKAREA	S	240	(mbrname1[,mbrname2] ... [,mbrnamen])	B210
XFERWKA	S	8	<i>membrane</i>	B210

STPROC Statement

This statement requests a call to a stored procedure program from another CA Telon program.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
DBINFO	BCIT	1	Y N Y=DBINFO flag is set	B268
IGNORE	BCIT	30	(cond1[,cond2] ... [,condn]) or ALL – list of return codes to ignore on return from a call to a stored procedure	S225
NAME	BCIT	8	Name of external stored procedure obtained from Catalog Extract	----
POSTSP	BCIT	8	cc point prior to call to stores procedure	S255
PRESPP	BCIT	8	cc point after call to stored procedure	S225
PRMSTYL	BCIT	1	D - DB2SQL, G - General, N - General with nulls, J - Java	B268
RESULTS	BCIT	2	Numeric - number of result sets being returned from stored procedure	B268
SPHDR	BCIT	5	Header of stored procedure being called	B100
SPID	BCIT	5	ID of store procedure being called	B100

* B = Batch: C = CICS: I = IMS: T = TSO

TABLE Statement

This statement describes the SQL table/view being referenced.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
COPY	BCITS	8	Specifies the copy member containing the I/O area structure layout	D141, D114
COPYLBL	BCITS	30	Specifies host variable to be used in references to the I/O area structure layout	D141, D114
COPYLV1	BCITS	1	Y N Y = copied I/O area structure begins at the "01 level"	D141, D114
DCLCOPY	BCITS	8	Copy member containing the DCLGEN layout for the table	D141
DCLLBL	BCITS	16	<i>hvnname</i> to be used in references to the DCLGEN structure	D141
DCLRDEF	BCITS	1	Y N Y = copy member is included in program working storage as a "redefines" of the DCLGEN area N = copy member is included within the standard I/O area of a CA Telon program	D141
DESC	BCITS	40	Description of table	D141
FROM	BCITS	240	Correlation names for table (QUAL, TABLE, Correlation Name)	D142
<i>label</i>	BCITS	8	Table identifier used in host name generation	D141
NAME	BCITS	18	Table name	D100, D141
QUAL	BCITS	8	Qualifier for table name	D100, D141
RDBMS	BCITS	8	Defines SQL type of table	D100
SYNONYM	BCITS	1	Y N Y = use SQL synonyms (table name will not be qualified) N = table is qualified	D141
TLNNAME	BCITS	8	CA Telon identifier	D141

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

TELON Statement

This statement identifies the version of CA Telon under which code is generated.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
APPLID	C	V	(CICS only) Application identifier	S112
HEADER	BCITS	5	Program header	S110, N110, S210, S310, B110
ID	BCITS	5	Program identifier	S110, N110, S210, S310, B110
LANG	BCITS	3	Program language (COB PLI)	S110, N110, S210, S310, B110
LANGLVL	BCITS	4	Current CA Telon level (5.0)	S110, N110, S210, S310, B110
LVLREQ	BCITS	1	Y N	

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

TLNJOIN Statement

This statement defines the DB2 tables/views to be referenced in a JOIN.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
FROM	BCITS	240	(Table qualifier, table name, correlation name) Defines the DB2 tables to be included in the join, as well as the correlation name to be associated with each table	D142
<i>label</i>	BCITS	8	Join label specifies the CA Telon identifier for the join to be used in <i>hvname</i> generation	D142, D143, D144
NAME	BCITS	18	Join name	D142
QUAL	BCITS	8	Qualifier for the join name	D142
RDBMS	BCITS	8	Defines table's SQL type	D142

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

TLNROW Statement

This statement permits multiple CA Telon views of a DB2 table/view. Each TLNROW permits different specifications of the CA Telon-defined column characteristics for each column in a DB2 table.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
<i>label</i>	BCITS	8	CA Telon identifier for the TLNROW to be used in the generation of USER I/O procedure or paragraph names	D141

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

TPPCB Statement

This statement identifies any teleprocessing PCBs that the program uses.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
ABCALL	BCIT	1	Y N Y = use generated PCB in place of XFER-PCB in ABEND processing	D211
EXPRESS	BIT	1	Y N Y = process messages sent to the PCB if an ABEND occurs	D211
LTERM	BIT	8	Logical terminal name	D211
MSGCALL	BIT	1	Y N Y = use this PCB in place of XFER-PCB for generated message switch processing	S127
<i>name</i>	BIT	12	<i>pcbname</i>	D211
<i>pcbname</i>	BIT	8	TPPCB name or type (e.g., XFER, IOPCB, ABEND)	D211
PRINT	BIT	1	Y N Y = use generated PCB in the print subroutine N = use XFER-PCB instead of the generated PCB	D211

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

TRANSACT Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LTHLBL	C	30	Overrides default generated COBOL or PL/I queue LTHOPT variable or record-level LTHLBL	S14Q

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

TSOPGM Statement

This statement requests generation of a TSO online program and defines its characteristics.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
GENPCBS	T	1	Y N Y = generate PCB masks N = include masks coded in LNKCOPY and USGCOPY members	F113
LNKCOPY	T	8	<i>mbrname</i>	F113
USGCOPY	T	8	<i>mbrname</i>	F113

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

UPDATE Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common Data Access parameters.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
LTHLBL	C	30	Overrides default generated COBOL or PL/I queue LTHOPT variable or record-level LTHLBL	S14Q

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

WORKSPA Statement

See [Data Access I/O Statement](#) (see page 424) for a list of common Data Access parameters.

XFEDIT Statement

This statement requests that, at program generation time, CA Telon creates code to perform a cross field validation and to handle any resulting errors.

Note: For all fields, commas or spaces can be used as separators.

Parameter	Env *BCITS	Size Bytes	Format/description	TDF Screens
COND	CIT	252	(cond1[,connector1,cond2] ... [,connectorn-1,condn]) Cond is <i>hvname</i> or operator combination. Operators are EQ, GE, GT, LT, LE, and NE. Connectors are AND, OR, and THENIF. For the two exceptions to the 253-byte length requirement, see the description following this table.	P165
CURSOR	CIT	8	<i>fldname</i>	P165
DESC	CIT	64	XFEDIT description	P161
ERRCHAR	CIT	60	(fldname1[,fldname2] ... [,fldnamen])	
ERRMSG	CIT	120	<i>hvname</i> 'literal' Error message	P165
HIGHLIGHT	CIT	120	(fldname1,fldname2] ... [,fldnamen])	P165
<i>label</i>	CIT	8	Edit name	P165
SEGLOOP	CIT	1	Y/N Indicates whether or not XFEDIT should be performed in a loop for a segloop field.	P165

* B = Batch; C = CICS; I = IMS; T = TSO, S = Stored Procedure

Exceptions to COND field

The following text describes two exceptions to the 252-byte length of the COND field:

1. The expanded or generated version of the edit condition in CA Telon format (as opposed to the CA Telon format code itself specified in the TDF) must not exceed 255 bytes ('IF' + 252 bytes).

For example:

EMPL-DOE, NE, XFER-CURRENT-DATE (29 bytes)

expands to

IF EMPL-DOE NOT = XFER-CURRENT-DATE (35 bytes)

2. COBOL or PL/I code in the edit condition that is to be carried literally (including enclosing single quotes) must not exceed 251 bytes.

For example:

'EMPL-DOE >XFER-CURRENT-DATE' (28 bytes)

expands to IF EMPL-DOE > XFER-CURRENT-DATE (31 bytes)

Procedural Custom Code

Custom code is any COBOL or PL/I statements that add additional processing logic to that which CA Telon automatically generates. You use such logic to perform processing beyond that which CA Telon normally performs (program-specific code).

Chapter "Custom Code" discusses custom code, its usage, and its purpose in detail.

The sample code labeled "Custom code in a COBOL program " shows how a custom code member might appear within CA Telon source. In this example, the **CONSIS** keyword on the SCREEN statement references the copy member **RMREQ**. The member itself is shown at the end of the screen definition. By definition, this copy member contains logic to be included in the consistency edits section of the program. **CONSIS** logic is always placed in the **X-100-CONSIS-EDITS** section, as illustrated by the example shown in the example labeled "Custom code in a COBOL program" and the example labeled "Custom code in PL/I program". The bottom of the COBOL example shows the part of the generated program that incorporates the **RMREQ CONSIS** logic.

As shown in the COBOL example, for each keyword that can identify custom code, there is a general purpose for the code referenced. There is also a general point, in the generated program, where CA Telon places the copy member code (see the second COBOL example).

CA Telon Source for a Screen Definition—COBOL

The following examples show the custom code in a COBOL program

Custom Code in a COBOL Program (Telon Source)

```

1 SCREEN MV10,CURSOR=OPTION,HEADER=MR,XFERWK=SSXFERW,      C
2 CONSIG=RMREQ,FLDEDIT=FLDEDIT,OINIT1=OINIT1,              C
3 PFKEYS=(1,2,3V,5,6,7,8,9V,12),SECTION=INITROOT,           C
4 WKAREA=WS,HELP=N,HOLD=N,ININT=ININIT,ALARM=Y,             C
5 OUTIFIL=UNDERLINE
6 DATA SET NAME=SSROOMS
7 SEGMENT OUTREAD,DBSEG=ROOM,SEGKEY=XFER-ROOM-ID,COPY=SSIR00T, C
8 KEYLEN=004,ACALL=NO
.
.
30 IMSPGM
31 END
.
.
34 ./ADD NAME=RMREQ
35 *****
36 * ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS *
37 * *
38 IF NEXT-PROGRAM-NAME-ID = 'MV20' OR 'MV30'
39 IF XFER-ROOM-ID = SPACES OR ERROR-REQ-CHAR
40 MOVE ERRRO-REQ-CHAR TO TPI-ID
41 MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
42 MOVE ERROR-ATTR TO TPO-ID-ATTR, TPO-OPTION-ATTR
43 MOVE '**ROOM ID REQUIRED FOR OPTION CHOSEN**' TO TPO-ERRMSG1
44 GO TO X-100-CONSIG-EDITS-RETURN.

```

Custom Code in Generated COBOL Program

```
IDENTIFICATION DIVISION.  
PROGRAM ID. MRCPMV10.  
.  
.  
X-100-CONSIG-EDITS SECTION.  
.  
.  
*TELON-----  
*  
*DS: MODLIB | COPY RMREQ  
*-----  
*  
*****  
* ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS *  
* *  
IF NEXT-PROGRAM-NAME-ID = 'MV20' OR 'MV30'  
.  
.  
.  
X-100-CONSIG-EDITS-RETURN.
```

CA Telon Source for a Screen Definition—PL/I

The following examples show the custom code in a PL/I program

Custom Code in a PL/I (Telon Source)

```

1 SCREEN PD10,CURSOR=OPTION,HEADER=MR,XFERWKA=SSIXFERW,      C
2 CONSIG=RMREQ,FLDEDIT=FLDEDIT,OINIT1=OINIT1,                C
3 PFKEYS=(1,2,3V,5,6,7,8,9V,12),SECTION=INITROOT             C
4 WKAREA=WS,HELP=N,HOLD=N,ININT=ININIT,ALARM=Y,              C
5 OUTIFIL=UNDERLINE
6 DATA SET NAME=SSROOMS
7 SEGMENT OUTREAD,DBSEG=ROOM,SEGKEY=XFER_ROOM_ID,COPY=SSIR00T, C
8 KEYLEN=004,ACALL=NO
.
.
30 IMSPGM
31 END
.
.
33 ./ADD NAME=RMREQ
34 /*****
35 / ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS *
36 /* *
37 IF NEXT_PROGRAM_NAME_ID = 'PD20'
38 NEXT_PROGRAM_NAME_ID = 'PD30'
39 THEN IF XFER_ROOM_ID = ' '
40 XFER_ROOM_ID = ERROR_REQ_CHAR
41 THEN DO;
42 TPI_ID = ERROR_REQ_CHAR;
43 CONTROL_INDICATOR = DO_WRITE_LIT;
44 TPO_ID_ATTR = ERROR_ATTR;
45 TPO_OPTION_ATTR = ERROR_ATTR;
46 TPO_ERRMSG1 = 'ROOM ID REQUIRED FOR OPTION CHOSEN';
47 GOTO X_100_CONSIG_EDITS_RETURN;
48 END;
```

Custom Code in Generated PL/I Program

```
WING CONSIS LOGIC MRPPD10: PROC
(DFHEIPTR,COMPRT) OPTIONS (MAIN REENTRANT);
.
.
X_100_CONSIS_EDITS: PROC;
.
.
/*TELON-----
/*DS: MODLIB %INCLUDE RMREQ
/*-----
/*****
* ROOM IS REQUIRED INPUT FOR ADD,DISP,UPDT, AND RMAV OPTIONS *
*/
IF NEXT_PROGRAM_NAME_ID = 'PD20'
NEXT_PROGRAM_NAME_ID = 'PD30'
.
.
X_100_CONSIS_EDIT_RETURN;
```

Generate Processing Flow

The generation process for TELON programs has several major steps, whether implemented on the mainframe with JCL procedures, or on the PWS with BAT or CMD files. These steps are discussed next.

Extraction of Exported Objects

The export feature of TELON produces a sequential file containing the program definition (named SCRNDEF) which contains macro invocations, followed by custom code members. These objects are preceded by IEBUPDTE separators of the form:

```
./ ADD NAME=nnnnnnnn
```

On the mainframe, the IEBUPDTE utility is used to load a temporary PDS with the objects. In PWS, the EXTRACT program performs a similar function, loading the objects as files in a directory. In each case, the SCRNDEF member is sent to the generator, while the custom code objects are set aside to await resolve processing (see below).

Preprocessing of SCRNDEF with ADPCCARD

The SCRNDEF member containing the TELON macro invocations is preprocessed by the ADPCCARD program which scans this member, adding the PGMTYPE macro invocation, and optionally overriding the final macro invocation(s) which cause the punching of generated code. The SYSIN file that serves as input to ADPCCARD is optional, and is usually specified as DUMMY. If it exists, the contents are used to replace the existing final macro invocation(s) which cause the punching of the program and/or screen maps. Thus, you can use it to generate programs for two different environments without having to re-export (as long as no other changes to the program or its custom code are required). For example, the SYSIN file might contain a new IMSPGM statement with the parameters necessary to generate for that environment, followed by an IMSMFS statement, both of which are used to replace an existing TSOPGM statement in the exported source.

Generation of Shell Program

The SCRNDEF member, containing modifications implemented by the ADPCCARD program is then passed to the assembler (on the mainframe) or the Assembler emulator Real390 (on the PWS) for the macro expansion process to create the program. There are a series of set-up macro invocations such as TELON, SCREEN, FIELD, and DATA SET which serve to populate the generator variable pool. Finally, the PGM macro (i.e., CICSPGM) is reached, and these variables are used to create the program, using the PUNCH feature of the macro assembler.

The punched output from the generator is produced as a sequential file with IEBUPDTE separators, with the generated shell program having the member name PROGRAM, and possibly other members are created containing link cards or other specifications. The PROGRAM member will contain a series of COPY (COBOL) or INCLUDE (PL/I) statements which will require resolution.

Extraction of Generated Objects

The sequential file containing the output of the generator is split into its component members with IEBUPDTE (on the mainframe) or the EXTRACT utility (on the PWS), with a temporary PDS or PWS directory loaded with the members.

Resolving of Custom Code

The PROGRAM member extracted in the previous step is then processed by the Resolve utility to insert the custom code members at the COPY or INCLUDE statements. Note that these custom code members may be obtained from both the original exported TELON objects and also from libraries. The procedures supplied insures that the exported TELON objects are searched prior to the libraries for resolution of the copies. Thus, if there are members with the same name in both the exported TELON source and in a library, the TELON source member will be copied.

At the conclusion of the resolve step, there should be a full COBOL or PL/I program available, with all COPYs or INCLUDEs accomplished. Depending on the environment and target platform, processing steps can be taken with the resulting program, such as a compile and link.

Chapter 14: Generated Working Storage Variables

CA-Telon generates many standard data division, file section, and working storage names. These data names, or variable names, are referred to in CA-Telon code. You may also want to refer to these data names in custom code.

This chapter describes the generated data names, and their usage. You will find yourself referring to this list when you are designing your custom code.

Alphabetical List of Field and Area Names

The following table lists field and area names used in CA-Telon- generated COBOL and PL/I programs.

All names used in both PL/I and COBOL applications have a hyphen (-) separator. Those fields appearing only in PL/I applications have an underscore (_) separator.

A one-letter code for the environment indicates that name is used in the indicated environment. The environments and their codes are:

The final column, Description, describes the field.

N - Nonterminal	I - IMS	R - Report	U - User defined
B - Batch	T - TSO	O - COBOL	L - CICS Character Client
C - CICS	S - CICS Character Server	D - Driver	P - PL/I
2 - Stored Procedure			

Environment	Generated Section Name	Description
N B C I T D R O P 2	ABT	Abnormal termination process. Use by the programmer is controlled by the installation.
N C O P 2	ABT-CJOURNAL-CICS-STAT US	Contains the EIB status byte after a CICS journal I/O has been performed.
N C O P 2	ABT-CQUEUE-CICS-STATUS	Contains the EIB status byte after a CICS queue I/O has been performed.

Environment	Generated Section Name	Description
B 2	ABT-DYNAMIC-CONTROL-R C	Field used to save the return code when ABNORMALT=3.
N C O P 2	ABT-ERROR-IS-CJOURNAL	COBOL 88-level used by the ABEND routine for journal activity.
N C O P 2	ABT-ERROR-IS-CQUEUE	COBOL 88-level used by the ABEND routine for queue activity.
B C I T D R O P 2	ADGADBER-ABEND-CODE	Field used to pass ABEND code to COBOL ABEND routine.
B C I T D R O P 2	ADGADBER-REASON-CODE	Field used to pass ABEND reason code to COBOL ABEND routine.
B I T D R P 2	ADGDBER_ABEND_CODE	Field used to pass ABEND code to PL/I ABEND routine.
B I T D R P IND\$FILE PUT SEC142 TXT A1 (ASCII CRLF RECFM V	ADGDBER_REASON_CODE	Field used to pass ABEND reason code to PL/I ABEND.
N B C I T D R O P 2	AIB	AIB block group name
N B C I T D R O P 2	AIBID	AIB block variable
N B C I T D R O P 2	AIBLEN	AIB block variable
N B C I T D R O P 2	AIBSFUNC	AIB block variable
N B C I T D R O P 2	AIBRSNM1	AIB block variable
N B C I T D R O P 2	AIBOALEN	AIB block variable
N B C I T D R O P 2	AIBOAUSE	AIB block variable
N B C I T D R O P 2	AIBRETRN	AIB block variable
N B C I T D R O P 2	AIBREASN	AIB block variable
N B C I T D R O P 2	AIBRSA1	AIB block variable
N B C I T D R O P 2	AIBPTR	AIB block variable
C R O	ALIAS-PROGRAM-NAME	Alias name of dynamic link module alternate entry point.
C I T R O P	BLANK-ATTR	Non-display, protect attribute byte.
C O P	BMSMAP-NAME	Literal name of the BMS map.
N B O P	BWA-ALL-DETAIL-PAGE-CO UNT	Count of all detail groups printed on current page.
N B O P	BWA-ALL-DETAIL-REPORT-	Count of all detail groups printed on report.

Environment	Generated Section Name	Description
	COUNT	
N B O P	BWA-AREA-LTH	Batch work area length.
N B O P	BWA-BLANK-WHEN-SAME-AREA	Work area used by BLANK-WHEN-SAME function.
N B O P	BWA-BOTTOM-DETAIL-LINE-NBR	Line number of last detail to be printed on a page.
N B O P	BWA-BWS-fldname	BLANK-WHEN-SAME compare field for <i>fldname</i> .
N B O P	BWA-BWS-grpname	BLANK-WHEN-SAME compare fields used by <i>grpname</i> .
N B O P	BWA-BWS-LITnn	BLANK-WHEN-SAME compare field for LITERAL nn.
C O P	BWA-filename-KEY	The generated key for the CICS VSAM file.
N O P	BWA-filename-LENGTH	The length of the spool record written to the CICS nonterminal VSAM file.
N O P	BWA-filename-RECORD	The field used to house the one line print buffer that is written to the CICS nonterminal VSAM file.
N B O P	BWA-fldname-TOTAL	Totaling field for <i>fldname</i> referenced by TOTREF parameter.
N B O P	BWA-grpname-COMPARE-VALUE	Current value of control break variable (CTLVAR).
N B O P	BWA-grpname-fldname-COUNT	Count of the number of times <i>fldname</i> in indicated <i>grpname</i> is printed.
N B O P	BWA-grpname-LAST-VALUE	Previous value of control break variable (CTLVAR).
N B O P	BWA-grpname-REPSEQ	Line-skip sequence used for indicated group.
N B O P	BWA-LAST-DETAIL	Last detail group processed.
N B O P	BWA-LAST-GROUP	Last group processed.
N B O P	BWA-LINE-COUNT	Number of lines printed on current page.
N B O P	BWA-LINES-TO-PRINT	Number of lines to be printed by current

Environment	Generated Section Name	Description
		group(s).
N B O P	BWA-MAJOR-CONTROL-GROUP	Field holding name of highest level CONTROL group.
N B O P	BWA-PAGE-BREAK-DETAIL	Field used to indicate when to print headings on new page.
N B O P	BWA-PAGE-BREAK-TYPE	Page break type: ENDPAGE DETAIL or CONTROL.
N B O P	BWA-PAGE-COUNT	Current number of pages printed on report.
N O P	BWA-PRINT-AREA	Data area used to write a CICS nonterminal report.
N B O P	BWA-PRINT-INDEX	Pointer used to build print lines into BWA-PRINT-AREA.
N O P	BWA-PRINT-LINE	The area used to build one logical page of output data for the nonterminal print routine.
N O P	BWA-PRINT-LINE-LENGTH	Contains report line length as specified on the TDF
N O P	BWA-PRINT-NEW-LENGTH	Initialized by the utility program ADLAENVL; used as the compressed length of the output buffer that is sent to the printer.
N O P	BWA-PRINT-ROWS	Contains the number of lines on a report page as specified on the TDF.
N O P	BWA-PRINTER-FF-CHAR	The printer form-feed attribute value used during the output of the print buffer.
N O P	BWA-PRINTER-ID	Contains the CICS print destination (printer ID) value.
N O P	BWA-PRINTER-NL-CHAR	The printer new-line attribute value used during the output of the print buffer.
N O P	BWA-PRINTER-STATUS	The internal status value used between the application program and the utility program ADLAPVER.
N O P	BWA-PRINTER-SCREEN-HEIGHT	The screen height for CICS nonterminal reports; value is determined with an EXEC CICS INQUIRE TERMINAL command and used to determine end of page for the nonterminal report.
N B O P	BWA-TRANSACTION-COUNT	Number of transactions that have been

Environment	Generated Section Name	Description
	T	processed.
NBCITDRO P 2	CATALOG-NAME	GET DIAGNOSTICS condition host variable
B O P	CHKP-FUNC	DL/I CHKP function code.
NBCITDRO P	CHNG-FUNC	DL/I CHNG function code
N C O P	CJOURNAL-filename-FIELD S	Group label for journal access.
N C O P	CJRNL-INVREQ	Indicates an invalid request for the journal operation.
N C O P	CJRNL-IOERR	Indicates that an I/O error occurred during the journal operation.
N C O P	CJRNL-JIDERR	Indicates that an invalid journal ID was used.
N C O P	CJRNL-LENGERR	Indicates that a length error occurred during the journal access.
N C O P	CJRNL-NOJBUFSP	Indicates that no journal buffer space has been detected.
N C O P	CJRNL-NOTAUTH	Indicates that invalid authorization was recognized during the journal operation.
N C O P	CJRNL-NOTOPEN	Indicates that the journal file/data set is not open.
N C O P	CJRNL-OK	Indicates that the journal operation was successful.
C I T D O	CLEAR	COBOL 88-level indicating clear key pressed: Value = 93.
B I T O P	CLSE-FUNC	GSAM CLSE function code
NBCITDRO P 2	CNTLERR-ABEND-CODE	Field holding U4001 ABEND issued for an undefined value of CONTROL-INDICATOR.
NBCITDRO P 2	CONDITION-NUMBER	GET DIAGNOSTICS condition host variable name
C I T R O P	CON SIS-INPUT-ERRORS	Generated only with PGMSTRUCT= 1 or 2. Checked at end of X-100 to determine processing flow (E = consistency edit error; ' ' = continue processing) in PGMSTRUCT 1 programs only.

Environment	Generated Section Name	Description
C I T R O	CONTINUE-PROCESS	COBOL 88-level indicating continue processing the transaction: Value = ' '.
C I T R O P	CONTINUE-PROCESS-LIT	Field holding ' ' to set CONTROL-INDICATOR to continue process.
N B C I T D R O P	CONTROL-INDICATOR	Process control indicator.
N B C I T D R	CONVERT-fldname-DB	The <i>hvname</i> (stored) values of CONVERT pairs for <i>fldname</i> .
N B C I T R O	CONVERT-fldname-INDEX	CONVERT table index for the <i>fldname</i> specified.
N B C I T R O	CONVERT-fldname-SCREEN	Screen image (displayed) values of CONVERT pairs for <i>fldname</i> .
N B C I T R O	CONVERT-TABLE-fldname	CONVERT table for the <i>fldname</i> .
N B C I T R O	CONVERT-TABLE-fldname-VALUES	CONVERT table values for the <i>fldname</i> .
N B C I T R O	CQUEUE-filename-FIELDS	Group label for this queue access.
N C O P	CQUEUE-IOERR	Indicates that an I/O error occurred during the queue operation.
N C O P	CQUEUE-ITEMERR	Indicates that the item specified could not be found (for READQ) or already exists (for WRITEQ).
N C O P	CQUEUE-ISCINVREQ	Indicates that an invalid request was specified for a queue that resides on a remote system.
N C O P	CQUEUE-LENGERR	Indicates that a length error occurred during the queue access.
N C O P	CQUEUE-NOTAUTH	Indicates that invalid authorization was recognized during the queue operation.
N C O P	CQUEUE-OK	Indicates that the queue READQ, WRITEQ, or DELETE Q was successful.
N C O P	CQUEUE-QBUSY	Indicates the error that can occur during a READQ TD queue operation when other I/O is pending against it.
N C O P	CQUEUE-QIDERR	Indicates that the TS or TD queue is not valid for the operation.
N C O P	CQUEUE-QZERO	Indicates that the READQ TD has reached end-of-file.
N C O P	CQUEUE-SYSIDERR	Indicates that an invalid SYSID was

Environment	Generated Section Name	Description
		specified with the queue operation or the system that the queue resides on is not operational.
N C O P	CQUEUE-TD-DISABLED	Indicates that the TD queue is disabled
N C O P	CQUEUE-TD-NOSPACE	Indicates that the TD queue has run out of space.
N C O P	CQUEUE-TD-NOTOPEN	Indicates that the TD queue is not open.
N C O P	CQUEUE-TS-NOTOPEN	Indicates that the TS queue is not open.
N B C I T D R O P 2	CURRENT-PROGRAM-NAME	Program name (header and ID) of current program
C I T O P	CURRENT-SEGMENT-KEY	First key on next page of list screen.
C I T R O P	CURSOR-ATTR	Position cursor, normal intensity, unprotect attribute byte
C I T R O P	CURSOR-BLANK-ATTR	Position cursor, non-display, unprotect attribute byte.
N B C I T D R O P 2	CURSOR-CLOSED-LIT	Field holding ' ' to set request-tblname CURSOR-IND to CURSOR-CLOSED.
N B C I T D R O P 2	CURSOR-NAME	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	CURSOR-OPEN-LIT	Field holding 'Y' to set request-tblname CURSOR-IND to CURSOR-OPEN.
B O P 2	DA-ALREADYOPEN	Indicates that the VSAM file is already open('41') (test is performed on file open)
B O P	DA-ALREADYOPEN-LIT	VSAM file ALREADY-OPEN literal
B O P 2	DA-ALREADYCLOSE	Indicates that the VSAM file is already closed ('42') (test is performed on file close)
B O P	DA-ALREADYCLOSE-LIT	VSAM file is ALREADY-CLOSED literal
N B C I T D R O 2	DA-ANYERROR	COBOL 88-level indicating any generic error STATUS.
N B C I T D R O 2	DA-DBMERROR	COBOL 88-level indicating any non-specific generic error has occurred.
N B C I T D R O P 2	DA-DBMERROR-LIT	Field holding 'DBM' to set DA-STATUS.
N C O P 2	DA-DISABLED	Indicates that the VSAM file has been disabled.

Environment	Generated Section Name	Description
N C O P 2	DA-DISABLED-LIT	VSAM DISABLED literal indicates that file attributes do not match the file ('39').
N C O P 2	DA-DUPKEY	Indicates that the VSAM duplicate key condition ('02') has been encountered
N C O P	DA-DUPKEY-LIT 2	VSAM DUPLICATE-KEY literal
N C O P 2	DA-DUPLICATE	Indicates that a VSAM duplicate key/record ('02' or '22') condition has been encountered
N C O P 2	DA-DUPLICATE-LIT	VSAM DUPLICATE-KEY/ DUPLICATE-RECORD literal.
N C O P 2	DA-DUPREC	Indicates that a VSAM duplicate record condition ('22') has been encountered
N C O P 2	DA-DUPREC-LIT	VSAM DUPLICATE-RECORD literal
B O P 2	DA-EMPTYFILE	Indicates that a VSAM file is empty ('34') (test is performed on file open)
B O P 2	DA-EMPTYFILE-LIT	Sequential file EMPTYFILE literal
N B C I T D R O 2	DA-ENDFILE	COBOL 88-level indicating an end-of-file has occurred.
N B C I T D R O 2	DA-ENDFILE-LIT	Field holding 'EOF' to set DA-STATUS.
N C O P	DA-INVREQ-LIT	Literal for the CJRNL-INVREQ condition.
N C O P	DA-ISCINVREQ-LIT	Literal to indicate that an invalid request was specified for a queue that resides on a remote system.
N C O P	DA-JIDERR	COBOL 88-level used to compare for the CJOURNAL-JIDERR condition.
N C O P	DA-JIDERR-LIT	Literal for the CJRNL-JIDERR condition.
N C O P	DA-JIOERR-LIT	Literal for the CJRNL-JIOERR condition.
N C O P	DA-JLENGERR-LIT	Literal for the CJRNL-JLENGERR condition.
N C O P	DA-JNOTAUTH-LIT	Literal for the CJRNL-JNOTAUTH condition.
N B C I T D R O P 2	DA-LOGICERR	COBOL 88-level indicating an invalid call sequence.
N B C I T D R O P 2	DA-LOGICERR-LIT	Field holding 'LOG' to set DA-STATUS.
N C O	DA-NOJBUFSP	COBOL 88-level used to compare for the CJOURNAL-NOJBUFSP condition.
N C O P	DA-NOJBUFSP-LIT	Literal for the CJRNL-NOJBUFSP condition.

Environment	Generated Section Name	Description
N C O P 2	DA-NOSPACE	Indicates that the VSAM file is out of space
N C O P 2	DA-NOSPACE-LIT	VSAM file OUT-OF-SPACE literal
N B C I T D R O P 2	DA-NOTAVAIL	COBOL 88-level indicating resource is unavailable.
N B C I T D R O P 2	DA-NOTAVAIL-LIT	Field holding 'NAV' to set DA-STATUS.
N B C I T D R O 2	DA-NOTFOUND	COBOL 88-level indicating a NOTFOUND condition.
N B C I T D R O P 2	DA-NOTFOUND-LIT	Field holding 'NFD' to set DA-STATUS.
N C O P	DA-NOTOPEN-LIT	Literal for the CJRNL-NOTOPEN condition.
N B C I T D R O 2	DA-OK	COBOL 88-level indicating a good return condition.
N B C I T D R O P 2	DA-OK-LIT	Field holding 'OK' to set DA-STATUS.
N C O	DA-QBUSY	COBOL 88-level used to compare for the DA-QBUSY condition.
N C O P	DA-QBUSY-LIT	Literal for the DA-QBUSY condition.
N C O	DA-QIDERR	COBOL 88-level used to compare for the DA-QIDERR condition.
N C O P	DA-QIDERR-LIT	Literal for the DA-QIDERR condition.
N C O	DA-QIOERR	COBOL 88-level used to compare for the DA-QIOERR condition.
N C O P	DA-QIOERR-LIT	Literal for the DA-QIOERR condition.
N C O P	DA-QLENGERR-LIT	Literal to indicate that a length error occurred during queue access.
N C O P	DA-QNOTAUTH-LIT	Literal to indicate that invalid authorization was recognized during the queue operation.
N C O	DA-QNTOPN-NOSPC	COBOL 88-level used to compare for the DA-QNTOPN-NOSPC condition.
N C O P	DA-NOPN-NOSPC-LIT	Literal for the DA-QNTOPN-NOSPC condition.
N C O P	DA-QZERO-LIT	Literal to indicate that the READQ TD call has reached end-of-file.
N B C I T D R O 2	DA-SECURITY	COBOL 88-level indicating a security violation has occurred.
N B C I T D R O P 2	DA-SECURITY-LIT	Field holding 'SEC' to set DA-STATUS.
N B C I T D R O P 2	DA-STATUS	Field holding generic return code.

Environment	Generated Section Name	Description
N C O P	DA-STATUS-CJOURNAL	The field used by the I/O routines to store the EIB response code for further application processing of journals.
N C O P	DA-STATUS-CQUEUE	The field used by the I/O routines to store the EIB response code for further application processing of queues.
N B C I T D R O P	DA-STATUS-DLI	Field holding DL/I return code.
N B C I T D R O P	DA-STATUS-FILE	Field holding VSAM or sequential return code.
N C O P	DA-SYSIDERR-LIT	Literal to indicate that an invalid sysid was specified or that the system that the queue resides on is not operational.
N C O	DA-TD-NOSPC	COBOL 88-level used to compare for the DA-TD-NOSPC condition.
N C O P	DA-TD-NOSPC-LIT	Literal for the DA-TD-NOSPC condition.
N C O P	DATASET-DISABLED	Field indicating data set is disabled in the FCT.
N C O P	DATASET-DSIDERR	Field indicating data set cannot be located in the FCT.
N B C O P	DATASET-DUPKEY	Field indicating multiple records found with duplicate alternate VSAM key.
N B C O P	DATASET-DUPREC	Field indicating attempt to add record with duplicate primary VSAM key.
N B C O P	DATASET-ENDFILE	Field indicating end-of-file condition during a browse (GETNEXT).
N B C O P	DATASET-ILLOGIC	Field indicating processing error not within other CICS/VS error class or logic error in batch file access.
N C O P	DATASET-INVREQ	Field indicating file access not permitted by data set entry in FCT.
N C O P	DATASET-IOERR	Field indicating data set I/O error code not in another CICS/VS error class.
N C O P	DATASET-ISCINVREQ	Field indicating remote system failure uncorrelated to a known condition.
N C O P	DATASET-LENGERR	Field indicating incorrect specification of the LENGTH option.
N C O P	DATASET-NOSPACE	Field indicating no direct access space is available for adding record.

Environment	Generated Section Name	Description
N C O P	DATASET-NOTAUTH	Field indicating resource security check has failed.
N B C O P	DATASET-NOTFND	Field indicating record not found in delete or retrieve attempt.
N C O P	DATASET-NOTOPEN	Field indicating requested data set is not open.
N B C O P	DATASET-OK	Field indicating no errors during file access.
N C O P	DATASET-SEGIDERR	Field indicating segment set not located in the FCT for this data set.
C O P	DATASET-SEQERR	Field indicating sequence error during VSAM file access.
N C O P	DATASET-SYSIDERR	Field indicating SYSID specified name not in table or system link closed.
N B C I T D R O P 2	DB2-AUTHENTICATION-TYPE	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-AUTHORIZATION-ID	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-COMMIT-CHANGE-IND	Indicates that a change access has occurred: Value = 'Y'.
N B C I T D R O P 2	DB2-COMMIT-READ-IND	Indicates that a read access has occurred: Value = 'Y'.
N B C I T D R O P 2	DB2-CONNECTION-STATE	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-CONNECTION-STATUS	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-ENCRYPTION-TYPE	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-ERROR-CODE1	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-ERROR-CODE2	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-ERROR-CODE3	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-ERROR-CODE4	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-GET-DIAGNOSTICS-DIAGS	GET DIAGNOSTICS statement host variable name

Environment	Generated Section Name	Description
N B C I T D R O P 2	DB2-INTERNAL-ERROR-POINTER	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-LAST-ROW	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-MESSAGE-ID	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-MODULE-DETECTING-ERROR	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-NUMBER-PARAMETER-MARKERS	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-NUMBER-RESULT-SETS	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-ORDINAL-TOKEN-&SCAL5	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-PRODUCT-ID	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-REASON-CODE	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-RETURN-STATUS	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-RETURNED-SQLCODE	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-ROW-NUMBER	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-SERVER-CLASS-NAME	GET DIAGNOSTICS connection host variable name
N B C I T D R O P 2	DB2-SQL-ATTR-CURSOR-HOLD	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-SQL-ATTR-CURSOR-ROWSET	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-SQL-ATTR-CURSOR-SCROLLABLE	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-SQL-ATTR-CURSOR-SENS	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-SQL-ATTR-CURSOR-TYPE	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	DB2-SQLERRD-SET	GET DIAGNOSTICS condition host variable

Environment	Generated Section Name	Description
		name
N B C I T D R O P 2	DB2-SQLERRD1	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-SQLERRD2	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-SQLERRD3	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-SQLERRD4	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-SQLERRD5	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-SQLERRD6	GET DIAGNOSTICS condition host variable name
N B C I T D R O P 2	DB2-TOKEN-COUNT	GET DIAGNOSTICS condition host variable name
N B C I T D R O P	DLET-FUNC	DL/I DLET function code
C O P	DLET-FUNC-WEIGHT	Single DL/I DLET call check- point weight factor
C O P	DLI-ACCUM-WEIGHT	accumulated checkpoint weight of all DL/I calls
N B C I T D R P	DLI_PARM_COUNT	Parm count field passed in PLITDLI calls.
C I T R O	DO-TRANSFER	COBOL 88-level indicating transfer control; Value = R.
C I T R O P	DO-TRANSFER-LIT	Field holding R to set CONTROL-INDICATOR to perform transfer.
C I T R O	DO-WRITE	COBOL 88-level indicating - send screen (may be an error iteration): Value = E.
C I T R O P	DO-WRITE-LIT	Field holding E to set CONTROL-INDICATOR to perform terminal write.
B O P	DSIO-ABEND-CODE	Field holding DSIO ABEND code issued for data set I/O error.
B O	dsname-EOF	COBOL 88-level indicating end of file was reached while reading a transaction: Value = 10.
B O	dsname-FILE	COBOL FILE SECTION FD file name.
N B O P	dsname-RECORD	Batch data set I/O area name.

Environment	Generated Section Name	Description
N B C O P	dsname-STATUS	Data set status after an attempted I/O.
B O P	END-LIT	Value of FILE-INDICATOR(n) when a MERGE file has encountered END-OF-FILE.
N B O	END-TRAN	COBOL 88-level indicating request to end transaction processing: Value = TRANSACTEND.
N B O P	END-TRAN-LIT	Field used to end transaction processing: Value = TRANSACTEND.
B O	ENDTRAN	COBOL 88-level indicating that the TRAN file has encountered END-OF-FILE (MATCH program only).
C I T D O	ENTER-KEY	COBOL 88-level indicating ENTER key pressed: Value = 0.
C I T R O P	ENTRY-CONTROL-INDICATOR	Value of CONTROL-INDICATOR on entry to transaction.
C I T R O	ENTRY-PROCESS-INPUT	COBOL 88-level indicating process input: Value = I.
C I T R O	ENTRY-PROCESS-OUTPUT	COBOL 88-level indicating process output: Value = O.
C I T O P U	ERROR-ATTR	Position cursor, high intensity, unprotect attribute byte.
C I T O P U	ERROR-MESSAGE-HELP	Field displayed on return from the HELP function.
C I T O P U	ERROR-MESSAGE-HIGHLIGHT	Field displayed when E-100 or J-100 edit errors occur.
C I T O P U	ERROR-MESSAGE-HOLD	Field displayed on return from the HOLD function.
C I T O P U	ERROR-MESSAGE-HOLD-ISRT	Field displayed when a HOLD is attempted with HOLD already active.
C I T O P U	ERROR-MESSAGE-MULTHIT	Field displayed when more than one SELECT option is chosen.
C I T O P U	ERROR-MESSAGE-NOHIT	Field displayed when no SELECT option is chosen.
C I T O P U	ERROR-MESSAGE-RESUME	Field displayed when no HOLD is active and RESUME attempted.
C I T O P U	ERROR-MESSAGE-SELECT-	Field displayed when invalid line number is

Environment	Generated Section Name	Description
	LINE-NO	selected during list processing.
C I T O P U	ERROR-REQ-CHAR	Character moved to required field when no data is entered by operator.
N B C I T D R O 2	FALLOUT-ABEND-CODE	Field holding U4002 ABEND code issued for improper section exit.
N B C I T D R O P	FIELD-EDIT-ERROR	Field edit return code.
N B C I T D R O	FIELD-EDIT-GOOD	COBOL 88-level field indicating successful edit: Value = ' '.
C O	FIELD-HAS-BEEN-MODIFIED	COBOL 88 indicating at least one screen field has been modified: Value = high-values.
N B C I T R O P	FIELD-INPUT-ERRORS	Generated only with PGMSTRUCT = 1 or 2. Checked at end of E-100 (or J-100 if SELECT fields) to determine processing flow (E or S = field edit error; ' ' = continue processing) for PGMSTRUCT=1 programs only.
N B O P	FIELD-LENGTH-TABLE	Table holding field lengths.
B O P	FILE-INDICATOR-TABLE	Merge input file status array.
B O P	FILE-INDICATOR(n)	Addressable entry within the FILE-INDICATOR- TABLE (Merge only).
N B O	FIRST-DETAIL	COBOL 88-level indicating first Detail group is being processed: Value = 1.
N B O	FIRST-TRAN	COBOL 88-level indicating first transaction is being processed: Value = 1.
B C I T D R O P	FLD-FUNC	DL/I FLD function code.
N B O P	FLT-fldname-LTH	FIELD-LENGTH-TABLE element holding length of <i>fldname</i> .
N B C I T R O P	FORMAT-fldname	Field holding the FORMAT for <i>fldname</i> .
N B C I T O P	FORMAT-fldname-LTH	Field holding the length of <i>fldname</i> .
N B C I T D R O P 2	GD-MORE	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	GD-NUMBER	GET DIAGNOSTICS statement host variable name
B O P	GET-LIT	Value of FILE-INDICATOR(n) when a READ is to take place on a particular MERGE file.
N B P	GET-TRAN	COBOL 88-level indicating request to input

Environment	Generated Section Name	Description
		transaction processing: Value = TRANSACTION.
N B O P	GET-TRAN-LIT	Field used to request an input transaction: Value = TRANSACTION.
B O P	GETTRAN-INDICATOR	TRAN file processing control (Match only).
B I T D R O P	GHN-FUNC	DL/I GHN function code.
N B I T D R O P	GHNP-FUNC	DL/I GHNP function code.
N B I T D R O P	GHU-FUNC	DL/I GHU function code.
N B I T D R O P	GN-FUNC	DL/I GN function code.
N B I T D R O P	GNP-FUNC	DL/I GNP function code.
B I T D R O P	GSAM-BLOCK-ID	Element of GSAM RSA.
B I T D R O P	GSAM-RECORD-DISP	Element of GSAM RSA.
B I T D R O P	GSAM-RSA	Group level GSAM record search argument (RSA).
B I T D R O P	GSAM-VOL-SEQ-NO	Element of GSAM RSA.
N B C I T D R O P	GU-FUNC	DL/I GU function code character to be entered by operator to request HELP Facility.
C I T O P U	HELP-CURR-MSG-COUNT	Table index key pointing to HELP message currently being displayed.
C I T O P U	HELP-FIELD-PGM-ID	Program name (header and ID) to get control to process field level Help.
C I T O P U	HELP-MSG-COUNT	Number of Help messages requested by operator.
C I T O P U	HELP-MSG-NAME	Table of Help message keys.
C I T O P U	HELP-SCREEN-PGM-ID	Program name (header and ID) to get control to process screen level Help.
C I T O P U	HELP-TABLE-LIMIT	Maximum number of field level Help requests allowed.
C I T R O P	HEX-3F	LINEOPT 2 value used for line traffic optimization.
C O P	HOLD-AREA-APPLID	3-byte ID used in key of the CICS TS queue for HOLD record.
C O P	HOLD-AREA-LTERM	Terminal ID used in key of CICS TS queue for HOLD record.

Environment	Generated Section Name	Description
C I T D O P	HOLD-AREA-TYPE	Set to D or P to indicate whether hold or help is requested.
B O P	HOLD-LIT	Value of FILE-INDICATOR(n) when a READ is to be bypassed on a particular MERGE file.
C I T D O P	HOLD-RESUME-PGM-ID	Program name (header and ID) to get control when Help or Hold processing has terminated.
C I T D O P	IBDX	Index to input buffer fields within a SEGLOOP.
C I T D O P	INPUT-BLANK-ATTR	Non-display, unprotect attribute byte.
C I T D O P	INPUT-HIGH-ATTR	High intensity, unprotect attribute byte.
N B C I T R O P	INPUT-LINE-COUNT	Current number of list input lines processed by E-100.
N B C I T R O P	INPUT-LINE-EDIT	Status of E-100 line-edit list iteration.
N B C I T R O P	IOA-segname-SEGMENT	Segment I/O area for specified <i>segname</i> .
N B C I T R O P	IOA-segname-SEGMENT-PA TH-n	Segment I/O area redefinition of UPDATE-AREA for PATH processing.
N B C I T R O P	IOA_segname_SEG_PTR	PL/I pointer to IOA_segname_SEGMENT.
B I T D R O P	IO-PCB	I/O PCB mask group label.
B O U	IOPCB-CHKP-ID	Batch exec DL/I only. Temporary storage for an exec DL/I CHKP or SYMCHKP call if ID is a literal.
B I T D R O P	IOPCB-JULIAN-DATE	Element of IO-PCB mask.
B I T D R O P	IOPCB-LTERM	Element of IO-PCB mask.
B I T D R O P	IOPCB-MOD-NAME	Element of IO-PCB mask.
B I T D R O P	IOPCB-MSG-SEQ	Element of IO-PCB mask.
B I T D R O P	IOPCB-STATUS-CODE	Element of IO-PCB mask.
B I T D R O P	IOPCB-TIME-OF-DAY	Element of IO-PCB mask.
B I T D R O P	IOPCB-USER-ID	Element of IO-PCB mask.
B O U	IOPCB-XRST-ID	Batch exec DL/I only. Temporary storage for the exec DL/I XRST call if id is a literal.
N B C I T D R O P	ISRT-FUNC	DL/I ISRT function code.
B O P	ISRT-FUNC-WEIGHT	single DL/I ISRT call checkpoint weight factor.

Environment	Generated Section Name	Description
C I T R O P U	LAST-LINENO	Number of iterations at end of list processing.
N B C I T D R O	LINE-ERRORS	COBOL 88-level indicating errors on current input list iteration edits: Value = E.
I D R O P 2	LINK-WORK-AREA	Common system fields passed from driver to static module.
I D O P	LINKAGE-OUTPUT-MODNAME	Name of IMS/DC MFS MOD used by a static program.
I D O P	LINKAGE-WRITE-XFER-INDIC	Next action to be performed by driver program.
B O P	MASTER-INDICATOR	MATCH key comparison result.
B O	MASTEREND	COBOL 88-level indicating that the MASTER file has encountered END-OF-FILE (MATCH program only).
B O	MASTERGET	COBOL 88-level indicating a read of the MASTER file must take place (MATCH program only).
B O P	MASTERHOLD	COBOL 88-level indicating a read of the MASTER file must be bypassed (MATCH program only).
B O P	MATCH-KEY-n	Addressable entry within the MATCH-KEY-SAVE-AREA.
B O P	MATCH-KEY-SAVE-AREA	Retains the prior records key from the transaction file.
B P	MAXSTOR	Used to contain the amount of storage remaining in the region to be used for the PL/I SORT programs.
B O P	MERGE-GROUP-n	Addressable entry within the MERGE-KEY-TABLE.
B O P	MERGE-KEY-TABLE	Retains keys of each of the MERGE input files.
N B C I T D R O P 2	MESSAGE-TEXT	GET DIAGNOSTICS condition host variable name
C I T O P	MORE-INDICATOR	Indicator controlling list screen forward paging. A value of "1" permits paging.
C I T O P U	MORE-LITERAL	Literal displayed on list screens to indicate there are more pages.
C I T D R O P	NEXT-PROGRAM-NAME	Program name (header and ID) of next

Environment	Generated Section Name	Description
		program to be executed.
C I T D R O P	NEXT-PROGRAM-NAME-ID	Screen ID of next program to be executed.
C O	NO-ALARM-ON-WRITE	COBOL 88-level indicating set screen alarm off: Value = low-values (X'00').
C O	NO-FIELD-MODIFIED	COBOL 88-level indicating no screen field modified: Value = low-values (X'00').
N B C I T R O	NO-LINE-EDIT	COBOL 88-level indicating no edits done on current input list iteration: Value = N.
N B C I T R O	NO-LINE-ERRORS	COBOL 88-level indicating no edit errors on current input list iteration: Value = ' '.
C I T R O P	OBDX	Index to output buffer fields within a SEGLOOP.
C I T R O P	OK-ATTR	Normal intensity, unprotect attribute byte.
N B C I T D R O P	OPEN-FUNC	GSAM OPEN function code.
N B C I T D R O P	OPERATOR-ID	Operator identifier.
C I T R O P	OUTPUT-ATTR	Normal intensity, protect attribute byte.
C I T R O P	OUTPUT-BLANK-ATTR	Non-display, protect attribute byte.
C I T R O P	OUTPUT-HIGH-ATTR	High intensity, protect attribute byte.
C I T D O	PA1-PA3	COBOL 88-levels indicating one of PA1-PA3 keys pressed: Values = 92,94,91.
C I T O P U	PAGE-AREA-START	Area used for automatic paging. Defined in transfer work area.
C I T R O	PAGE-BACKWARD	COBOL 88-level indicating request for backward paging: Value = 2.
C I T R O	PAGE-FORWARD	COBOL 88-level indicating request for forward paging: Value = 1.
C I T O P	PAGE-KEY-1	First key in PAGE-KEY-TABLE.
C I T O P	PAGE-KEY-2-END	All keys except first key in PAGE-KEY-TABLE.
C I T O P	PAGE-KEY-SAVE	Redefinition of PAGE-KEY-TABLE with multiple occurrences.
C I T O P	PAGE-KEY-TABLE	Tables of keys to pages already displayed by list.
C I T O P	PAGE-NUMBER-SAVE	Current page number being displayed by list screen.

Environment	Generated Section Name	Description
C I T O P	PAGE-REQUEST-INDICATOR	Control field for a paging request.
C I T O P	PAGE-SAVE-MAX	Maximum number of pages to be saved (taken from SEGLOOP parameter PAGESAV).
C I T O P	PAGE-WORK-AREA	Redefinition of PAGE-AREA-START.
C I T O P	PAGES-MAVED	Count of pages being held in PAGE-KEY-TABLE.
N C O P	PCB-FUNC	DL/I PCB schedule request.
N B C I T D R O P U	PCB-LIST	Area holding pointers to PCBs used in the program.
N B C I T D R O P U	Pcbname-DBD-NAME	Element of PCB mask.
N B C I T D R O P U	Pcbname-KEY-FB-AREA	Element of PCB mask.
N B C I T D R O P U	Pcbname-LENGTH-FB-KEY	Element of PCB mask.
N B C I T D R O P U	Pcbname-NO-SENS-SEGS	Element of PCB mask.
N B C I T D R O P U	Pcbname-PCB	Label generated or user defined for a database PCB mask.
N B C I T D R O P U	Pcbname-PROC-OPTIONS	Element of PCB mask.
N B C I T D R O P U	Pcbname-RECORD-LENGTH	Element of PCB mask.
N B C I T D R O P U	Pcbname-SEG-LENGTH	Element of PCB mask for GSAM operations.
N B C I T D R O P	Pcbname-segname-1	First byte of pcbname-STATUS-CODE used to test generic DL/I status codes.
N B C I T D R O P	Pcbname-segname-STATUS	Value of last DL/I status code for specified <i>segname</i> .
N B C I T D R O P U	Pcbname-STATUS-CODE	Element of PCB mask.
C I T D O	PFK1-PFK12	COBOL 88-level indicating PF1 through PF12 key pressed: Values = 1-12.
C I T D O	PFK1-12-PFK13-24	COBOL 88-levels indicating PF1 through PF24 pressed: Values = 1-24
C I T R O P	PFKEY-INDICATOR	Field to indicate which PF key was struck.
C I T R O P	PFKEY-RETURN-INDICATOR	Generated only with PGMSTRUCT = 1 or 2. Checked at end of P-100 to determine processing flow (E = PF key error processing; R = transfer control; ' ' = continue processing) for PGMSTRUCT = 1 programs only.

Environment	Generated Section Name	Description
N B C I T D R O P	POS-FUNC	DL/I POS function code.
R O P	PRINT-LTERM-NAME	LTERM of destination for online IMS/DC report.
I R O P	PRINT-PURG	DL/I purge function.
N B O	PRINT-PURGE-INDICATOR	Control indicator to invoke PURGE call for IMS/DC report.
N B R O P	PROCESS-detail	COBOL 88-level field(s) set to value of each of the <i>grpnames</i> in the batch program.
C I T R O	PROCESS-grpname-LIT	Field used to request detail group processing: Value = <i>grpname</i> .
C I T R O	PROCESS-INPUT	COBOL 88-level indicating perform MAIN-INPUT: Value = I.
C I T R O P	PROCESS-INPUT-LIT	Field holding I to set CONTROL-INDICATOR to perform input process.
C I T R O	PROCESS-OUTPUT	COBOL 88-level indicating perform MAIN-OUTPUT: Value = O.
C I T R O P	PROCESS-OUTPUT-LIT	Field holding O to set CONTROL-INDICATOR to perform output process.
N B C I T D R O P 2	PROGRAM-NAME	ID of the current program being executed.
N B C I T 2	PROGRAM-TYPE	Program type: ■ B = batch If SQL is active: ■ C = CICS 2 = Batch ■ I = IMS 2 = TSO ■ T = TSO.
C I T R O P	PROT-ATTR	Normal intensity, protect attribute type.
C O P	PSB-NAME	Name of PSB scheduled for this program.
B O P	PURG-FUNC	DL/I PURGE function code.
N B C I T D R O P	REPL-FUNC	DL/I REPL function code.
B O P	REPL-FUNC-WEIGHT	Single DL/I REPL call check-point weight factor.
N B C I T D R O P 2	Request-tblname-CURSOR-IND	Indicates whether the CURSOR defined by the specified request for the given <i>tblname</i> is open or closed.
N B C I T D R O P 2	request-tblname-CURSOR-OPEN	COBOL 88-level indicating CURSOR for request/tblname is open: Value = 'Y'.

Environment	Generated Section Name	Description
N B C I T D R O P 2	RETURNED-SQLSTATE	GET DIAGNOSTICS condition host variable name
N B O P	RFL-fldname	Name of single field in the printline RFL-grpname-LINE-n.
N B O P	RFL-fldname-CHAR	Output field character definition of RFL-fldname when PIC is used.
N B O P	RFL-grpname-LINE-n	Entire nth print line of <i>grpname</i> .
N B O P	RFL-LITnn	Literal field name used when BLANK-WHEN-SAME is coded.
N B C I T D R O P	ROLB-FUNC	DL/I ROLB function code.
N B C I T D R O P	ROLL-FUNC	DL/I ROLL function code.
N B C I T D R O P 2	ROW-COUNT	GET DIAGNOSTICS statement host variable name
N B C I T D R O P 2	SAVE-GETDIAG-SQLCODE	GET DIAGNOSTICS variable used to save SQLCODE
C O P	SCI-ALARM-INDICATOR	Indicates whether or not alarm is to be rung for this screen.
C O P	SCI-MODIFY-INDICATOR	Indicates if any field on screen has been modified.
C O P	SCI-WRITE-INDICATOR	Indicates if screen has previously been written.
C O	SCREEN-FIRST-WRITE	COBOL 88-level indicating screen not yet written: Value = low values (X'00').
C O	SCREEN-HAS-BEEN-WRITT EN	COBOL 88-level indicating screen has been written: Value = high values (X'FF').
C I T R O P	SCREEN-IMAGE	Current screen values held in the transfer work area.
N B C I T D R O P	SEC-INDEX	Index to SECTION-NAME-TABLE trace table.
N B C I T D R O P 2	SECTION-NAME-TABLE	Trace table of sections being executed.
C I T O P	SEGLOOP-1-SSA	Unqualified SSA for initial DL/I GU and GN calls for an output list Group level mask.
C I T R O P	SEGLOOP-1-SSA-CMDCODE	Element of SEGLOOP <i>start-browse</i> SSA mask.
C I T R O P	SEGLOOP-1-SSA-IMSKEY	Element of SEGLOOP <i>start-browse</i> SSA mask.

Environment	Generated Section Name	Description
C I T R O P	SEGLOOP-1-SSA-LPAREN	Element of SEGLOOP <i>start-browse</i> SSA mask.
C I T R O P	SEGLOOP-1-SSA-OPCODE	Element of SEGLOOP <i>start-browse</i> SSA mask.
C I T R O P	SEGLOOP-1-SSA-SEGMENT	Element of SEGLOOP <i>start-browse</i> SSA mask.
C I T R O P	SEGLOOP-1-SSA-VALUE	Element of SEGLOOP <i>start-browse</i> SSA mask.
C I T O P	SEGLOOP-2-SSA	Qualified SSA for start browse processing in list screens.
C I T R O P	SEGLOOP-2-SSA-CMDCODE	Element of multi-page SEGLOOP SSA mask.
C I T R O P	SEGLOOP-2-SSA-IMSKEY	Element of multi-page SEGLOOP SSA mask.
C I T R O P	SEGLOOP-2-SSA-LPAREN	Element of multi-page SEGLOOP SSA mask.
C I T R O P	SEGLOOP-2-SSA-OPCODE	Element of multi-page SEGLOOP SSA mask.
C I T R O P	SEGLOOP-2-SSA-SEGMENT	Element of multi-page SEGLOOP SSA mask.
C I T R O P	SEGLOOP-2-SSA-VALUE	Element of multi-page SEGLOOP SSA mask.
N B C I T R O P	SEGLOOP-COUNT	Current iteration in B-100 output list processing.
C I T R O P	SEGLOOP-COUNT-MAX	Maximum iterations allowed for list processing.
C I T O P	SEGLOOP-ERROR-SW	Switch for tracking errors in SEGEDITS or XFEDITS performed in a loop for segloop fields.
N B C I T R O P	segname-altssa-SSA	User requested alternate SSA group level mask.
N B C I T D R O P	segname-altssa-SSA-CMDCODE	Element of ALTSSA mask.
N B C I T D R O P	segname-altssa-SSA-SEGMENT	Element of ALTSSA mask.
N B C R O P	segname-ddname	Constant with value <i>ddname</i> used for debugging.
N B C I T R O P	segname-QUAL-SSA	Qualified default alternate SSA name for <i>segname</i> .
N B C I T R O P	segname-SSA	SSA generated for <i>segname</i> based on segment usage. Group level SSA mask.
N B C I T D R O P	segname-SSA-CMDCODE	Element of SSA mask.

Environment	Generated Section Name	Description
N B C I T D R O P	segname-SSA-IMSKEY	Element of SSA mask.
N B C I T D R O P	segname-SSA-LPAREN	Element of SSA mask.
N B C I T D R O P	segname-SSA-OPCODE	Element of SSA mask.
N B C I T D R O P	segname-SSA-SEGMENT	Element of SSA mask.
N B C I T D R O P	segname-SSA-VALUE	Element of SSA mask.
N B C I T R O P	segname-UNQUAL-SSA	Unqualified default SSA name for sequence.
N B C I T D R O P 2	SERVER-NAME	GET DIAGNOSTICS condition host variable name
C O	SET-ALARM-ON-WRITE	COBOL 88-level indicating set screen alarm on: Value = high-values (X'FF')
B P	SORT-RETURN	Used to contain the RETURN-CODE passed back at the end of the sort.
N B C I T R O P	SOUND-THE-ALARM	Used to set the IMS MFS SCA field to ring the alarm.
N C I T D R O P	SPA-AREA	DFHCOMMAREA, WORKSPA, or SPA transfer work area structure.
N C I T D O P	SPA-TRANSACTION-CODE	Field in the SPA-AREA that holds the next transaction. Initialized to current transaction on program entry.
N C I T D R O P	SPA-XFER-WORK-AREA	Application defined transfer work area in the SPA-AREA.
C I T O P	START-BROWSE-KEY	Transfer area variable name generated when programmer uses SEGLOOP STBRKEY parameter.
N B C I T D R O P	SYNC-FUNC	DL/I SYNC function code.
N C O P	SYSWK-filename-JOURNAL-LENGTH	The length of logical record that will be written.
N C O P	SYSWK-filename-JOURNAL-PFXLEN	Field that contains the generated prefix length.
N C O P	SYSWK-filename-JOURNAL-PREFIX	System default area generated for a journal prefix.
N C O P	SYSWK-filename-JOURNAL-REQID	System default area generated for the journal request ID.
N C O P	SYSWK-filename-QUEUE-ITEM	The halfword system default area used for the TS ITEM parameter.

Environment	Generated Section Name	Description
N C O P	SYSWK-filename-QUEUE-LENGTH	The length of logical record that will be read or written.
N C O P	SYSWK-filename-QUEUE-NAME	Name of the TS or TD to be accessed.
N C O P	SYSWK-filename-QUEUE-NITEM	The halfword system default area used for the TS NUMITEM parameter.
N C O P	SYSWK-filename-QUEUE-PNTR	The fullword area that will be used as a system default for the TS SET= parameter.
N C O P	SYSWK-filename-QUEUE-SYSID	Contains the CICS queue SYSID name.
N B C I T D R O P 2	Tblname-colname-NN	Generated NOT NULL indicator for <i>colname</i> in <i>tblname</i> : Value = '0'.
N B C I T D R O 2	Tblname-colname-NU	COBOL 88-level indicating NULL value for <i>colname</i> : Value = '1'.
N B C I T D R O P 2	TELON-ABNORMALT-FEATURE	Indicates whether the program was generated with abnormal termination (ABNORMAL) type 1,2 or 3
N B C I T D R O 2	TELON-COBOL-VERSION	Version for COBOL for which program was generated
N B C I T B R O 2 P	TELON-CREATE-DATE	Date the program was created in (or imported to) the TDF. Generated only if GENDTES is set to 'Y' in either the SETSYS or environment SETENV in the TLNIIS used to generate the program.
N B C I T D R O P 2	TELON-EATTR-FEATURE	Indicates whether program was generated with extended attributes (EATTR) on (Y) or off (N)
N B C I T D R O P 2	TELON-GEN-DATE	System date on which the program was generated
N B C I T D R O P 2	TELON-GEN-TIME	System time at which the program was generated
N B C I T D R O P 2	TELON-LINEOPT-FEATURE	Indicates whether program was generated with LINEOPT 1, 2, 3
N B C I T D R O P 2	TELON-MOD-DATE	Current CA-Telon mod date.
N B C I T D R O P 2	TELON-MOD-NO	Current CA-Telon mod number.
N B C I T D R O P 2	TELON-PGM-ID	CA-Telon mod identifier.
N B C I T D R O P 2	TELON-PGMSRTUCT-FEATURE	Indicates whether the program was generated with PGMSTRUCT 1, 2 or 3

Environment	Generated Section Name	Description
N B C I T D R P 2	TELON-PLI-VERSION	Version of PL/I for which program was generated
N B C I T D R O P 2	TELON-REL-DATE	Current CA-Telon release date.
N B C I T D R O 2	TELON-REL-MOD-ID	Current CA-Telon release level.
N B C I T D R O P 2	TELON-RELEASE-EYECATCH	"TELON-ID"
N B C I T D R O P 2	TELON-TRACE-OPTION	Indicates whether CA-Telon tracing is turned on (Y) or off (N)
N B C I T D R O P 2	TELON-UPDATE-DATE	Date the program was last updated in TDF. Generated only if GENDTES is set to 'Y' in either the SETSYS or environment SETENV in the TLNIIS used to generate the program.
N B C I T D R O P 2	TELON-UPDATE-TIME	Time the program was last updated in TDF. Generated only if GENDTES is set to 'Y' in either the SETSYS or environment SETENV in the TLNIIS used to generate the program.
N B C I T D R O P 2	TELON-UPDATE-USER	User the program was last updated in TDF. Generated only if GENDTES is set to 'Y' in either the SETSYS or environment SETENV in the TLNIIS used to generate the program.
N C O P	TERM-FUNC	DL/I TERM function code.
T R O P	TP-OUTPUT-MODNAME	Name of IMS MFS MOD for this program.
C I T O P	TPI-fldname	Name of screen field in the input buffer.
C I T O	TPI-fldname-HELP	Redefinition of TPI-fldname to allow help request check.
C I T O P	TPI-fldname-LTH	Length of <i>fldname</i> as defined in the panel image.
I O P	TPI-IOINDIC	Indicator to select output or input processing.
I T R O P	TPI-PFKEY	Indicates which PF key has been pressed.
C I T R O P	TPO-ERRMSG1	Name of error message field in the output buffer.
C I T R O P	TPO-fldname	Name of screen field in the output buffer.
C I T R O P	TPO-fldname-ATTR	Name of attribute byte in output buffer for <i>fldname</i> .

Environment	Generated Section Name	Description
C I T R O	TPO-fldname-CHAR	Output field definition of TPO-fldname when PIC is used.
C I T R O P	TPO-fldname-LTH	Length of the field.
B I T D R O P	tppcb-LTERM	Element of TPPCB mask.
B I T D R O P	tppcb-PCB	Group level of TPPCB mask.
B I T D R O P	tppcb-STATUS-CODE	Element of TPPCB mask.
N B C I T D R O P 2	TRACE-FIELD-NAME	Last field edited in output or input processing. Generated only when TRACE is set to 'Y' in the program environment in exported source or the TLNIIS used to generate the program.
N B C I T D R O P 2	TRACE-SECTION-NAME	Name of the program section currently being executed. Generated only when TRACE is set to 'Y' in the program environment in exported source or the TLNIIS used to generate the program.
N B C I T D R O P 2	TRACE-SEGMENT-NAME	Name of last DL/I segment, VSAM, or sequential file or table accessed. Generated only when TRACE is set to 'Y' in the program environment in exported source or the TLNIIS used to generate the program.
B O P	TRAN-COUNTER	Counts the number of transactions that have a matching master record (MATCH program only).
B O P	TRAN-INDICATOR	TRAN file read indicator (Match only).
B O	TRANDONE	COBOL 88-level indicating the end of a group of transactions that had matching keys (MATCH program only).
B O	TRANGET	COBOL 88-level indicating a read of the TRAN file must take place (MATCH program only).
B O	TRANHOLD	COBOL 88-level indicating a read of the TRAN file must be bypassed (MATCH program only).
B O	TRANREAD	COBOL 88-level indicating a read of the TRAN file must take place (MATCH program only).
C I T O P	TRANSACTION-COMPLETE	COBOL 88-level indicating terminate transaction: Value = C.

Environment	Generated Section Name	Description
C I T R O P	TRANSACTION-COMPLETE-LIT	Field holding C to set CONTROL-INDICATOR to terminate transaction.
N B C I T D R O P U	UPDATE-AREA	Buffer used in GHU call of combination GHU-REPL calls.
N C O P	UPDATE-PTR	Address of retrieved record set on VSAM READ for UPDATE.
N B C I T D R O P 2	WORKFLD-ALPHA	Used by editing to pass alphanumeric data to/from screen fields.
N B C I T D R P 2	WORKFLD_BIT	Used by editing to pass bit data to/from screen fields.
N B C I T D R O P 2	WORKFLD-INDEX	General purpose, binary half-word work field.
N B C I T D R O P 2	WORKFLD-LTH	Used by editing to pass length of variable length data to/from screen fields.
N B C I T D R O P 2	WORKFLD-NUMERIC	Used by editing to pass numeric data to/from screen fields.
N B C I T D R O P 2	WORKFLD-NUMERIC-n	Alternate form of WORKFLD-NUMERIC for specified edits.
N C O P	WORKFLD-NUMREC	Number of records deleted by CICS VSAM GENERIC DELETE.
N C O P	WORKFLD-SEGLTH	CICS VSAM segment length.
N B C I T D R O P 2	WORKFLD-VCHAR	Used by editing to pass alphanumeric part of variable length data to/from screen fields.
C I T O P U	XFER-HOLD-INDICATOR	Value P = HELP/HOLD or Value D = HOLD only processing has occurred.
B I T D R O P	XFER-PCB	IMS/DC TPPCB to perform automatic message switching. Group level for XFER-PCB mask.
B I T D R O P	XFERPCB-LTERM	Element of XFER-PCB mask.
B I T D R O P	XFERPCB-STATUS-CODE	Element of XFER-PCB mask.
B O P	XRST-FUNC	DL/I XRST function code.

Section and Procedure Names

This subject lists the COBOL sections and PL/I procedures that CA-Telon generates for both online and batch programs. To find out when various sections or procedures are generated, see the Program Hierarchical Charts in Chapter 15.

Each section/procedure in a list is preceded by columns identifying the environment in which it is used. The online section/procedure names are first, followed by the batch names (see "Batch Programs").

Online Programs

C - CICS	I - IMS	T - TSO
D - DRIVER	R - Report	O - COBOL
P - PL/I	S - CICS Character Server	L - CICS Character Client

Environment	Generated Section Name	Description
D O P	A-100-DRIVER-INIT	Driver initialization
S C I T R O P	A-100-OUTPUT-INIT	Output initialization
S C I T R O P	B-100-OUTPUT-EDITS	Output field edits
S O	B-100-OUTPUT-EDIT- <i>fldname</i>	Format field <i>fldname</i> for output
S C I T R O P	B-100-OUTPUT-SEGLOOP-EDITS	Output list field edits
S C I T R O P	B-100-OUTPUT-SEGLOOP-EXIT	Exit list loop
C I T R S L O P	B-100-OUTPUT-SEGLOOP-GET-NEXT	Get next record for BROWSE
S C I T R O	B-100-OUTPUT-SEGLOOP-INIT	Initialize list processing
S O	C-100-CLIENT-READ	Receive data from client
C O P	C-100-MERGE-IN	Merge input message
L C I D O P	C-100-TERMIO-READ	Receive input message
L C O	C-100-TERMIO-RECEIVE	Receive input message
S O	C-200-CLIENT-WRITE	Prepare to return data to client
L C I T D R O P	C-200-TERMIO-WRITE	Write screen
C O P	C-210-TERMIO-WRITE-INITIAL	Initialize screen write
C L O P	C-220-TERMIO-WRITE-SUBSEQUEN T	Perform subsequent writes to screen

Environment	Generated Section Name	Description
L C I T D O P	C-300-TERMIO-XFER	Transfer to next program
D O	C-310-CALL-pgmname	Process CA-Telon dynamic link
I D O P	C-400-TERMIO-XFER-MSG-SWITCH	Message switch processing
L O	C-500-FORM-INIT	Perform form initialization processing
L O	C-600-PROCESS-FORM	Perform form processing
L O	C-700-TP-TO-CLI-ATTR	Map attribute settings
L O	C-710-TP-SEARCH-TABLE	Search for TP attribute values
L O	C-800-CLI-TO-TP-ATTR	Map attribute values
L O	C-810-CLI-SEARCH-TABLE	Search for client attribute values
I D O P	C-900-GET-SPA	Get IMS SPA
D O	C-900-GET-WORKSPA	Read WORKSPA database
I D O P	C-910-GET-MESSAGE	Get input message
I D O P	C-920-GET-WORKSPA	Read WORKSPA database
L C I R O P	C-930-INPUT-MERGE	Merge screen image with input buffer
L C I R O P	C-940-OUTPUT-MERGE	Merge screen image with output buffer
I D O P	C-950-PUT-WORKSPA	Write WORKSPA database
D O P	C-960-PUT-SPA	Write SPA
I D R O P	C-970-PUT-MESSAGE	Write message
S O	C-980-ATTRIB-INIT	Initialize screen attributes
R O P	C-980-SET-DESTINATION	Set print destination
S O	C-990-BUFFER-INIT	Initialize output buffer
T O P	D-100-DRIVER-TERM	Terminate driver main line
S C I T O P	D-100-INPUT-INIT	Input initialization
C I T R S L O P	E-100-INPUT-CUSTOM-CODE	Contains ICUST2 custom code
S L C I T O P	E-100-INPUT-EDITS	Input field edits
S O	E-100-INPUT-EDIT- <i>fldname</i>	Validate <i>fldname</i> on input
C I T R S L O P	E-100-INPUT-SEGLOOP	Validate segloop input fields
C I T R S L O P	E-100-INPUT-SEGLOOP-INIT	Setup for input segloop processing
C I T R S L O P	E-100-INPUT-SEGLOOP-END	Contains FLDEDIT custom code
S O	E-200-PROCESS-FIELD	Process input fields

Environment	Generated Section Name	Description
SCITOP	H-100-INPUT-TERM	Input termination
SCITDROP	G-100-GET-DIAGNOSTICS	Get diagnostics
IRO	IMS-ALTERNATE-ENTRY	IMS program secondary entry section
IRO	IMS-PRIMARY-ENTRY	IMS program main process entry section
IRO	IMS-PRIMARY-ENTRY-PROCESS	IMS program main process transaction section
SLCITOP	J-100-SELECT	Select field processing
SLCITOP	J-100-SELECT-OPT-fldname	Processing for <i>fldname</i> .
LCITOP	K-100-HOLD-RESTORE	Restore from HOLD
LCOP	K-200-HOLD-NOTFND	HOLD restore not processed
SLCOP	K-200-HOLD-RESUME	Return from HOLD
LCITOP	K-200-RESUME-OK	Process HOLD restore
LCITOP	L-100-HOLD-ERROR	Error in HOLD processing
SLCITOP	L-100-HOLD-SAVE	Store current information
LCITOP	L-100-OK-TO-HOLD	Process HOLD transfer
CITDRO	MAIN	COBOL main line processing
SO	MAIN-FIELD-PROCESS	Process fields
SO	MAIN-FORM-INIT	Main process for form initialization
SO	MAIN-FORM-PROCESS	Process form
SO	MAIN-FORM-TERM	Termination for form processing
LCITOP	MAIN-INPUT	Input processing
SLCITOP	MAIN-LINE	Main line processing
LCITDROP	MAIN-OUTPUT	Output processing
SLCITO	MAIN-PROCESS	Determine initial processing
CITDROP	MAIN-PROCESS-LOOP	Main processing loop
CITDRP	MAIN-SECTION	PL/I main line processing
LCITOP	M-100-HELP-ANALYZE	Check for field level HELP
SLCITOP	N-100-CURSOR-POSITION	Position cursor
SLCITOP	P-100-PFKEYS	PF key processing
SLCOP	Q-100-CICS-INIT	Program load initialization
COP	Q-200-PSB-SCHEDULE	Schedule PSB

Environment	Generated Section Name	Description
C O P	Q-210-PSB-TERM	Terminate schedule PSB
S O	S-100-SERVER-TERM	Server processing termination
C I T R O P	S-100-STP-CALLS	Stored procedure calls
C I T R O P	S-100-CALL-storedprocedure	Call stored procedure
C I T R O P	S-200-STP-CURSORS	Cursor processing for called stored procedure result sets.
C I T R O P	U-100-request-segname	Access requested file
C I T R O P	U-100-USER-IO	User I/O processing
S C I T R O P	X-100-CONSIS-EDITS	Consistency edits processing
S C I T R L O P	Z-100-SECTIONS-COPY	SECTION custom code
R O P	Z-900-PROGRAM-END	Program end
S L C I T D O P	Z-900-SECTION-FALLOUT	Fallout of section catch
C I T R S L O P	Z-980-ABNORMAL-TERM	Perform abnormal termination process
S L C I T D O P	Z-990-PROGRAM-ERROR	Invalid control value catch.

Batch Programs

A - Match Structure	B - Standard Structure	E - Merge Structure
O - COBOL	P - PL/I	S - Main Sort Structure

Environment	Generated Section Name	Description
A O P	A-1000-MAST-EQ-TRAN	Match transaction-equals-master processing
A O P	A-1000-MAST-GREATER	Match master-greater processing
A O P	A-1000-MATCH	Match processing
B S E O P	A-1000-PROCESS-TRAN	Process transaction
A O P	A-1000-TRAN-GREATER	Match transaction-greater processing
B S A E O P	B-1000-PROCESS-DETAIL	Process detail groups
B S A E O P	B-2000-END-REPORT	Report termination
B S A E O P	B-5000-FORMAT-grpname	Format <i>grpname</i> detail for print

Environment	Generated Section Name	Description
B S A E O P	B-6000-PRINT-grpname	Print <i>grpname</i> detail
B S A E O P	B-9000-PAGE-BREAK	Process page break
E O P	C-1000-COMPARE-KEYS	Merge key comparison processing
A O P	C-1000-GET-MAST	Get Master file record
B S A E O P	C-1000-GET-TRAN	Get transaction
E O P	C-1000-SET-INDICATORS	Set file indicators for Merge processing
A O P	C-1000-TRAN-DONE	Transaction processing complete
A O P	C-1000-TRAN-PROCESS	Transaction record processing
B S A E O P	C-2000-WRITE-REPORT	Write report
B S A E O P	G-100-GET-DIAGNOSTICS	Get diagnostics
B S A E O	MAIN	COBOL main line processing
S O P	MAIN-OUTPUT-LOOP	Main program processing
B A E O P	MAIN-PROCESS-LOOP	Main processing loop
B S A E P	MAIN_SECTION	PL/I main line processing
S O P	MAIN-SORT-OUTPUT	Main sort processing
B S A E O P	Q-1000-PROGRAM-INIT	Program initialization
O P	R-1000-PARSE-PARM	Parse input parameter(s)
B S A E O P	S-100-STP-CALLS	Stored procedure calls.
B S A E O P	S-100-call-storedprocedure	Call stored procedure
B S A E O P	S-200-STP-CURSORS	Cursor processing for called stored procedures result sets
O P	S-1000-sortname	Processing for sort <i>sortname</i>
O P	S-1000-sortname-INPUT	Input sort processing for sort <i>sortname</i>
O P	S-1000-sortname-OUTPUT	Output sort processing for sort <i>sortname</i>
O P	S-1000-USER-SORT	User sort processing
B S A E O P	T-1000-PROGRAM-TERM	Terminate processing
B S A E O P	U-100-request-segname	Access requested file, table, etc.
B S A E O	U-100-USER-IO	User I/O processing
B S A E O	Z-900-SECTION-FALLOUT	Fall out of section

Environment	Generated Section Name	Description
B S A E O P	Z-980-ABNORMAL-TERM	Term perform abnormal termination processing
B S A E O P	Z-990-PROGRAM-ERROR	Invalid control value

CICS Nonterminal Programs

O - COBOL	P - PL/I

Environment	Generated Section Name	Description
O P	A-N100-PROCESS-TRAN	Process transaction
O P	B-N100-PROCESS DETAIL	Process detail group
O P	B-N200-END-REPORT	Report termination
O P	B-N500-FORMAT-grpname	Format grpname detail for print
O P	B-N600-PRINT-grpname	Print grpname detail
O P	B-N900-PAGE-BREAK	Process page break
O P	C-N100-GET-TRAN	Get transaction
O P	C-N200-WRITE-REPORT	Write report
O P	G-100-GET-DIAGNOSTICS	Get diagnostics
O	MAIN	COBOL main line processing
O P	MAIN-PROCESS-LOOP	Main processing loop
P	MAIN_SECTION	PL/I main line processing
O P	Q-N100-PROGRAM-INT	Program initialization
O P	Q-N200-PSB-SCHEDULE	Schedule PSB
O P	Q-N210-PSB-TERM	Terminate PSB
O P	Q-N300-ACQUIRE-WKAREAS	Acquire work areas
O P	S-100-STP-CALLS	Stored procedure calls
O P	S-100-CALL-storedprocedure	Call stored procedure
O P	S-200-STP-CURSORS	Cursor processing for called stored procedure result sets
O P	T-N100-PROGRAM-TERM	Program termination

Environment	Generated Section Name	Description
O P	U-100-request-segname	Access requested file, table, etc.
O	U-100-USER-IO	User I/O processing

Stored Procedure Programs

O - COBOL	P - PL/I	2 - Stored Procedure

Environment	Generated Section Name	Description
2 O P	A-1000-STORED-INIT	Preliminary processing and parameter mapping
2 O P	C-3000-STORED-PROCESS	Main processing for stored procedure
2 O P	D-1000-STORED-TERM	Terminating processing and parameter mapping
2 O P	G-100-GET-DIAGNOSTICS	Get diagnostics
2 O	MAIN	COBOL main line processing
2 O P	PROCESS-LOOP	Main Processing loop
2 O P	Q-1000-PROGRAM-INIT	Program initialization
2 O P	T-1000-PROGRAM-TERM	Terminate processing
2 O P	U-100-USER-IO	User I/O processing
2 O P	U-100-request-segname	Access requested table
2 O P	Z-100-SECTIONS-COPY	Location for SECTION custom code
2 O P	Z-900-SECTION-FALLOUT	Fall out of section catch
2 O P	Z-980-ABNORMAL-TERM	Term perform abnormal termination processing
2 O P	Z-990-PROGRAM-ERROR	Invalid control catch value

PF Key Variables

When you press a PF/PA key, CA-Telon sets the **PFKEY-INDICATOR** to one of the following COBOL 88-level values.

```
88 ENTER-KEY VALUE 00.  
88 CLEAR VALUE 93.  
88 PA1 VALUE 92.  
88 PA2 VALUE 94.  
88 PA3 VALUE 91.  
  
88 PFK1 VALUE 1. 88 PFK1-13 VALUE 1 13.  
88 PFK2 VALUE 2. 88 PFK2-14 VALUE 2 14.  
88 PFK3 VALUE 3. 88 PFK3-15 VALUE 3 15.  
88 PFK4 VALUE 4. 88 PFK4-16 VALUE 4 16.  
88 PFK5 VALUE 5. 88 PFK5-17 VALUE 5 17.  
88 PFK6 VALUE 6. 88 PFK6-18 VALUE 6 18.  
88 PFK7 VALUE 7. 88 PFK7-19 VALUE 7 19.  
88 PFK8 VALUE 8. 88 PFK8-20 VALUE 8 20.  
88 PFK9 VALUE 9. 88 PFK9-21 VALUE 9 21.  
88 PFK10 VALUE 10. 88 PFK10-22 VALUE 10 22.  
88 PFK11 VALUE 11. 88 PFK11-23 VALUE 11 23.  
88 PFK12 VALUE 12. 88 PFK12-24 VALUE 12 24.
```

DL/I Area Layouts

The following example shows a PCB mask created for a CA-Telon IMS or CICS application using DL/I database access. Note that **pcbname** is either the **PCBNAME** parameter on the Update DBMS Type screen or the **DBDNAME** parameter on the Create/Update Data Group screen.

COBOL

```
01 pcbname-PCB.  
  05 pcbname-DBD-NAME PIC X(8).  
  05 pcbname-SEG-LEVEL PIC X(2).  
  05 pcbname-STATUS-CODE PIC X(2).  
  05 pcbname-PROC-OPTIONS PIC X(4).  
  05 FILLER PIC X(4).  
  05 pcbname-SEG-NAME-FB PIC X(8).  
  05 pcbname-LENGTH-FB-KEY PIC S9(5) COMP.  
  05 pcbname-NO-SENS-SEGS PIC S9(5) COMP.  
  05 pcbname-KEY-FB-AREA PIC X(n).
```

PL/I

```

DCL pcbname_PCB_PTR POINTER;
DCL 1 pcbname_PCB BASED(pcbname_PCB_PTR),
    05 pcbname_DBD_NAME CHAR(8),
    05 pcbname_SEG_LEVEL CHAR(2),
    05 pcbname_STATUS_CODE CHAR(2),
    05 pcbname_PROC_OPTIONS CHAR(4),
    05 FILL1 CHAR(4),
    05 pcbname_SEG_NAME_FB CHAR(8),
    05 pcbname_LENGTH_FB_KEY FIXED BIN(31),
    05 pcbname_NO_SENS_SEGS FIXED BIN(31),
    05 pcbname_KEY_FB_AREA CHAR(n);

```

The following DL/I mask is used for GSAM access when the DATA SET TYPE is set to GSAM.

COBOL

```

01 pcbname-PCB.
   05 pcbname-DBD-NAME PIC X(8).
   05 pcbname-SEG-LEVEL PIC X(2).
   05 pcbname-STATUS-CODE PIC X(2).
   05 pcbname-PROC-OPTIONS PIC X(4).
   05 FILLER PIC X(4).
   05 pcbname-SEG-NAME-FB PIC X(8).
   05 pcbname-LENGTH-FB-KEY PIC S9(5) COMP.
   05 pcbname-NO-SENS-SEGS PIC S9(5) COMP.
   05 pcbname-KEY-FB-AREA PIC X(8).
   05 pcbname-RECORD-LENGTH PIC X(4).

```

PL/I

```

DCL pcbname_PCB_PTR POINTER;
DCL 1 pcbname_PCB BASED(pcbname_PCB_PTR),
    05 pcbname_DBD_NAME CHAR(8),
    05 pcbname_SEG_LEVEL CHAR(2),
    05 pcbname_STATUS_CODE CHAR(2),
    05 pcbname_PROC_OPTIONS CHAR(4),
    05 FILL1 CHAR(4),
    05 pcbname_SEG_NAME_FB CHAR(8),
    05 pcbname_LENGTH_FB_KEY FIXED BIN(31),
    05 pcbname_NO_SENS_SEGS FIXED BIN(31),
    05 pcbname_KEY_FB_AREA CHAR(8),
    05 pcbname_RECORD_LENGTH CHAR(4);

```

An IMS/DC I/O PCB defined by a TP PCB (specified on the Create/Update Data Group screen) is as follows:

COBOL

```
01 IO-PCB.  
  05 IOPCB-LTERM PIC X(8).  
  05 FILLER PIC X(2).  
  05 IOPCB-STATUS-CODE PIC X(2).  
  05 IOPCB-JULIAN-DATE PIC S9(7) COMP-3.  
  05 IOPCB-TIME-OF-DAY PIC S9(7) COMP-3.  
  05 IOPCB-MSG-SEQ PIC S9(7) COMP.  
  05 IOPCB-MOD-NAME PIC X(8).  
  05 IOPCB-USER-NAME PIC X(8).
```

If the TLNIIS (SETSYS or SETENV) parameters IOPCBM is set to "Y", the following additional two fields (supported by IMS v6 and above) are generated:

```
05 IOPCB-GROUP-NAME PIC X(8)  
05 IOPCB-TIMESTAMP PIC X(12)
```

PL/I

```
DCL IO_PCB_PTR POINTER;  
DCL 1 IO_PCB BASED(IO_PCB_PTR),  
  05 IOPCB_LTERM CHAR(8),  
  05 FILLER CHAR(2),  
  05 IOPCB_STATUS_CODE CHAR(2),  
  05 IOPCB_JULIAN_DATE FIXED DEC(7),  
  05 IOPCB_TIME_OF_DAY FIXED DEC(7),  
  05 IOPCB_MSG_SEQ FIXED BIN(31),  
  05 IOPCB_MOD_NAME CHAR(8),  
  05 IOPCB_USER_ID CHAR(8);
```

If the TLNIIS (SETSYS or SETENV) parameter IOPCBM is set to "Y", the following additional fields (supported by IMS v6 and above) are generated:

```
05 IOPCB_GROUP_NAME CHAR(8),  
05 IOPCB_TIMESTAMP CHAR(12);
```

For generated message switches, CA-Telon uses an IMS/DC alternate TP PCB that is structured as follows:

COBOL

```
01 XFER-PCB.  
  05 XFERPCB-LTERM PIC X(8).  
  05 FILLER PIC X(2).  
  05 XFERPFB-STATUS-CODE PIC X(2).  
  
DCL XFER_PCB_PTR POINTER;  
DCL 1 XFER_PCB BASED(XFER_PCB_PTR),  
  05 XFERPCB_LTERM CHAR(8),  
  05 FILLER CHAR(2),  
  05 XFERPCB_STATUS_CODE CHAR(2);
```

An IMS/DC alternate TP PCB defined by a TPPCB (specified on the Create/Update Data Group screen) is structured as follows:

COBOL

```
01 tppcb-PCB.  
  05 tppcb-LTERM PIC X(8).  
  05 FILLER PIC X(2).  
  05 tppcb-STATUS-CODE PIC X(2).
```

PL/I

```
DCL tppcb_PCB_PTR POINTER;  
DCL 1 tppcb_PCB BASED(tppcb_PCB_PTR),  
  05 tppcb_LTERM CHAR(8),  
  05 FILLER CHAR(2),  
  05 tppcb_STATUS_CODE CHAR(2);
```

DL/I SSA Layouts

The segname-dscname-SSA is the segment search argument (SSA) generated for the named segment. The segname-dscname-SSA is either the LABEL or the DBSEG parameter on the Create/Update Data Group and the Update Database Segment screens. The segname-dscname-SSA is generated in the following form:

COBOL

```
05 segname-dscname-SSA.  
  10 segname-dscname-SSA-SEGMENT PIC X(8) Value 'DC'.  
  10 segname-dscname-SSA-CMDCODE PIC X(4) Value '*---'.  
  10 segname-dscname-SSA-LPAREN PIC X(1) Value '('.  
  10 segname-dscname-SSA-IMSKEYn PIC X(8) Value 'DCF1'.  
  10 segname-dscname-SSA-OPCODEn PIC X(2) Value '='.  
  10 segname-dscname-SSA-VALUEn PIC X(7).  
  10 segname-dscname-SSA-BOOLOPn PIC X(1) Value '#'.  
  10 FILLER PIC XX.
```

PL/I

```
DCL 1 segname_dscname_SSA,  
  05 segname_dscname_SSA_SEGMENT CHAR(8),  
  05 segname_dscname_SSA_CMDCODE CHAR(4),  
  05 segname_dscname_SSA_LPAREN CHAR(1),  
  05 segname_dscname_SSA_IMSKEYn CHAR(8),  
  05 segname_dscname_SSA_OPCODEn CHAR(2),  
  05 segname_dscname_SSA_VALUEn CHAR(n),  
  05 segname_dscname_SSA_BOOLOPn CHAR(1),  
  05 FILL1 CHAR(2);
```

(Where dscname is eliminated in the case that dscname = segname.)

DL/I RSA Layouts

The GSAM-RSA is the record search argument (RSA) generated for GSAM access.

COBOL

```
01 GSAM-RSA.  
  10 GSAM-BLOCK-ID PIC X(4).  
  10 GSAM-VOL-SEQ-NO PIC X(2).  
  10 GSAM-RECORD-DISP PIC X(2).
```


PL/I

```
DCL 1 GSAM_RSA,  
    05 GSAM_BLOCK_ID FIXED BIN (31),  
    05 GSAM_VOL_SEQ_NO FIXED BIN (15),  
    05 GSAM_RECORD_DISP FIXED BIN (15),
```

There are three different SEGLOOP SSA layouts depending on the database operation being performed.

SEGLOOP-1-SSA is the SSA generated for the initial (page 1) GU call and for subsequent GN calls for the page for a segment defined with a BROWSE usage.

If the STBRKEY parameter is not specified on the Create/Update File SEGLOOP screen, the layout of the generated SSA is:

COBOL

```
05 SEGLOOP-1-SSA.  
    10 SEGLOOP-1-SSA-SEGMENT PIC X(8).  
    10 SEGLOOP-1-SSA-CMDCODE PIC X(4).
```

PL/I

```
DCL 1 SEGLOOP-1-SSA,  
    05 SEGLOOP_1_SSA_SEGMENT CHAR(8),  
    05 SEGLOOP_1_SSA_CMDCODE CHAR(5);
```

If the **STBRKEY** parameter is specified on the SEGLOOP, the layout of the generated SSA is:

COBOL

```
05 SEGLOOP-1-SSA.  
    10 SEGLOOP-1-SSA_SEGMENT PIC X(8).  
    10 SEGLOOP-1-SSA_CMDCODE PIC X(4).  
    10 SEGLOOP-1-SSA_LPAREN PIC X.  
    10 SEGLOOP-1-SSA_IMSKEY PIC X(8).  
    10 SEGLOOP-1-SSA_CMDCODE PIC XX.  
    10 SEGLOOP-1-SSA_VALUE PIC X(n).  
    10 FILLER PIC XX.
```

PL/I

```
DCL 1 SEGLLOOP_1_SSA,  
    05 SEGLLOOP_1_SSA_SEGMENT CHAR(8),  
    05 SEGLLOOP_1_SSA_CMDCODE CHAR(4),  
    05 SEGLLOOP_1_SSA_LPAREN CHAR(1),  
    05 SEGLLOOP_1_SSA_SCHKEY CHAR(8),  
    05 SEGLLOOP_1_SSA_OPCODE CHAR(2),  
    05 SEGLLOOP_1_SSA_VALUE CHAR(n),  
    05 FILL1 CHAR(2);
```

SEGLLOOP-2-SSA is the SSA generated for the GU (Get Unique) call used to establish database position in CA-Telon paging screens other than the first page. The layout of the generated SSA is:

COBOL

```
05 SEGLLOOP-2-SSA.  
    10 SEGLLOOP-2-SSA-SEGMENT PIC X(8).  
    10 SEGLLOOP-2-SSA-CMDCODE PIC X(4).  
    10 SEGLLOOP-2-SSA-LPAREN PIC X.  
    10 SEGLLOOP-2-SSA-IMSKEY PIC X(8).  
    10 SEGLLOOP-2-SSA-CMDCODE PIC XX.  
    10 SEGLLOOP-2-SSA-VALUE PIC X(n).  
    10 FILLER PIC XX.
```

PL/I

```
DCL 1 SEGLLOOP_2_SSA,  
    05 SEGLLOOP_2_SSA_SEGMENT CHAR(8),  
    05 SEGLLOOP_2_SSA_CMDCODE CHAR(4);  
    05 SEGLLOOP_2_SSA_LPAREN CHAR(1);  
    05 SEGLLOOP_2_SSA_UNQKEY CHAR(8);  
    05 SEGLLOOP_2_SSA_OPCODE CHAR(2);  
    05 SEGLLOOP_2_SSA_VALUE CHAR(n);  
    05 FILL1 CHAR(2);
```

DL/I Linkage

The PCB argument list used in the generated application is shown below.

GENPCBS=N is generated when the **CICS**PGM or **BATCH**PGM parameter is specified. When **GENPCBS=Y**, you must include this same list in the hhPROC copy member. The following illustration shows the argument list generated for the CA-Telon training database.

COBOL

```
IO-PCB  
XFER-PCB  
ABEND-PCB  
MPRINT-PCB  
LPRINT-PCB  
EMPLOYEE-PCB  
HELP-PCB  
HOLD-PCB  
WORKSPA-PCB
```

PL/I

```
IO_PCB_PTR,  
XFER_PCB_PTR,  
ABEND_PCB_PTR,  
MPRINT_PCB_PTR,  
LPRINT_PCB_PTR,  
EMPLOYEE_PCB_PTR,  
HELP_PCB_PTR,  
HOLD_PCB_PTR,  
WORKSPA_PCB_PTR;
```


Chapter 15: Advanced TDF Concepts

This chapter further discusses how to use the CA Telon Design Facility to develop batch and online applications. It does not cover CA Telon issues relating to the specific target transaction processors (CICS or IMS), batch environments or the underlying database management systems (DB2, IMS, and such). The appendixes cover these issues.

This chapter covers:

- Using BASE definitions
- SEGLOOPing into a table with paging
- Alternative output mapping in a SEGLOOP
- Output SEGLOOP browsing
- Consistency edits in SEGLOOPS
- Referencing screen attributes in PL/I
- Individual field edit error messages
- The 3270 light pen feature
- Minimizing the size of SPA/COMMAREAs
- Line optimization and screen merge processing
- CA Telon screen handling
- Using the configuration section
- 3270 numeric lock feature
- The CA Telon HELP facility

Using Base Definitions

Individual CA Telon screen programs in an application system or subsystem frequently contain a number of characteristics that are global to the system or subsystem. The headings or footings on each screen, the list of databases or data sets accessed, and the IMS or CICS environment definitions are often similar, if not identical. Ideally, you should define and test information that is consistent across a system only once. All screen programs should use the proven information as a model. Model CA Telon screen or batch programs are referred to as *BASE* definitions.

BASE Panel Definitions

BASE Panel Definitions (PDs) usually define the headings and footings you want on every screen in the system. Typically, these can contain such items as:

- The system's screen title(s)
- Date and time fields
- The error message field(s)
- Transaction and/or key data fields
- Literals defining the system's use of PF keys

You should use the BASE panel(s) for any type of globally used image information. Once you merge a BASE panel with a particular application panel for the system, you can make changes. The BASE is not necessarily a rigid model; it is a starting point to encourage standards and to lessen the amount of duplicate development required across a system.

To implement a BASE PD, develop individual screen programs' PDs with empty areas in them for the field information brought in as the BASE PD. Use the COPY command to merge the BASE PD with a screen program's PD. Or use the CA Telon TDF Utilities to copy the BASE PD first, and then update the *new* PD to include the panel specific fields, and so on.

High Level Base Definitions

The next level of BASING is at the screen/report/driver/batch/stored procedure (SD/RD/DR/BD/SP) level. These high-level *BASE* definitions can:

- Include system-wide PF-key code (SD only)
- Include the transfer work area copy code (SD/ND/RD/DR only)
- Include other standard pieces of copy code (for example WKAREAs) or SECTIONS
- Standardize the OUTIFIL character (SD only)
- Standardize the extended attribute defaults (SD only)
- Standardize labelling for PCBNAMES, segments, and user execs within base data group definitions
- Define standard file or database accesses within a base data group (for example, accesses to the HOLD database, the WORKSPA database, or other system-wide files).
- Define IMS/CICS/batch/stored procedure environment

Creating and Using BASE Definitions

To create a BASE, first create a base panel image and panel definition as described earlier in this guide. Then, create an SD, RD, DR, BD, or SP to contain the BASE characteristics.

To use the BASE SD/ND/RD/DR/BD/SP to create the individual application's SD/ND/RD/DR/BD/SP, enter the Header-id combination for the base in the BASE name area of the Option 4 or Option 5 sub-menu.

SEGLOOPing into a Table with Paging

In this example, you must accumulate multiple input screens of data in a table (array) before you can update any files.

This type of routine is useful for batch balancing, where the totals from the data entered must balance to some pre-determined amount. If an out-of-balance condition occurs, the operator must page backward and forward through the table to find the amount(s) in error. This example uses PF8 to page forward, PF7 to page backward, and an input field on the screen (MORE) to indicate that more data must be entered. If the operator presses Enter, the processing continues to the H-100-INPUT-TERM section for balancing and updating of the files.

This example shows only the COBOL custom code to control the paging and to output blank screens so you can enter more data. The following topics describe this code.

Panel definition

In the Panel Definition (PD), the fields are OUTIN and mapped from/to a table in the transfer work area. This table is subscripted by SEGLOOP-COUNT plus a counter (CTR). SEGLOOP-COUNT is automatically incremented in the B-100-OUTPUT-SEGLOOP and E-100-INPUT-SEGLOOP sections of the program, and the counter divides the table into pages of data.

```
010 DBNAME='XFER-TABLE-FIELD (CTR)'
```

OINIT1 custom code

Enter the code, shown below, into the A-100 OINIT1 custom code entry point.

```
020 IF XFER-FIRST-TIME-INDR IS EQUAL TO 'Y'  
030 MOVE 'N' TO XFER-FIRST-TIME-INDR  
040 MOVE 1 TO XFER-PAGE-NUMBER  
050 MOVE ZERO TO XFER-MAX-PAGES.  
060 IF XFER-PAGING-INDICATOR IS EQUAL TO 'F'  
070 ADD 1 TO XFER-PAGE-NUMBER.  
080 IF XFER-PAGING-INDICATOR IS EQUAL TO 'B'  
090 SUBTRACT 1 FROM XFER-PAGE-NUMBER.  
100 MOVE SPACE TO XFER-PAGING-INDICATOR.
```

STATEMENTS 020-050, on the first time through, initialize the first time indicator, paging forward/backward indicator, current page number, and maximum number of pages built so far.

STATEMENTS 060-070 add one to the current page number if you want to page forward.

STATEMENTS 080-100 subtract one from the current page number if you want to page backward.

OCUST1 custom code

Enter the code, shown below, into the B-100 OCUST1 custom code entry point.

```
110 COMPUTE CTR = ((SEGLLOOP-COUNT-MAX * XFER-PAGE-NUMBER) -  
120 SEGLLOOP-COUNT-MAX) + SEGLLOOP-COUNT.  
130 IF XFER-PAGE-NUMBER IS GREATER THAN XFER-MAX-PAGES  
140 MOVE XFER-PAGE-NUMBER TO XFER-MAX-PAGES  
150 GO TO B-100-OUTPUT-SEGLLOOP-EXIT.
```

STATEMENTS 110-120 compute the counter used to divide the table into pages of data.

STATEMENTS 130-150 output a blank screen so you can enter more data if the current page number is greater than the maximum number of pages built so far.

OCUST2 custom code

Enter the code, as shown below, into the B-100 OCUST2 custom code entry point.

```
160 ADD 1 TO CTR.
```


CONSIS custom code

Enter the code, as shown below, into the X-100 CONSID custom code entry point.

```
170 IF TPI-MORE IS EQUAL TO 'Y'
180 MOVE 'F' TO XFER-PAGING-INDICATOR
190 MOVE XFER-MAX-PAGES TO XFER-PAGE-NUMBER
200 MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
210 GO TO X-100-CONSID-EDITS-RETURN.
220 IF PFKEY-INDICATOR IS EQUAL TO 8
230 AND XFER-PAGE-NUMBER IS LESS THAN XFER-MAX-PAGES
240 MOVE 'F' TO XFER-PAGING-INDICATOR
250 MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
260 GO TO X-100-CONSID-EDITS-RETURN.
270 IF PFKEY-INDICATOR IS EQUAL TO 8
280 MOVE SPACE TO XFER-PAGING-INDICATOR
290 MOVE 'CANNOT PAGE FORWARD - NO MORE PAGES'
300 TO TPO-ERRMSG1
310 MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
320 GO TO X-100-CONSID-EDITS-RETURN.
330 IF PFKEY-INDICATOR IS EQUAL TO 7
340 AND XFER-PAGE-NUMBER IS GREATER THAN 1
350 MOVE 'B' TO XFER-PAGING-INDICATOR
360 MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
370 GO TO X-100-CONSID-EDITS-RETURN.
380 IF PFKEY-INDICATOR IS EQUAL TO 7
390 MOVE SPACE TO XFER-PAGING-INDICATOR
400 MOVE 'CANNOT PAGE BACKWARD FROM PAGE 1'
410 TO TPO-ERRMSG1
420 MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
430 GO TO X-100-CONSID-EDITS-RETURN.
```

STATEMENTS 170-210 set the paging forward/backward indicator to forward, set the current page number to the maximum number of pages built so far, and transfer control to the output side of the program. This allows you to enter more data.

STATEMENTS 220-260 set the paging forward/backward indicator to forward and transfer control to the output side of the program when you page forward and there are more pages.

STATEMENTS 270-320 rewrite the screen with an error message when you page forward and there are no pages.

STATEMENTS 330-370 set the paging forward/backward indicator to backward and transfer control to the output side of the program when you page backward and are not at page one.

STATEMENT 380-430 rewrite the screen with an error message when you page backward and are at page one.

Alternative Output Mapping in a SEGLOOP

When CA Telon maps data to a screen defined to include columns of data, the sequence of the data is left to right, top to bottom.

```
value 1 value 2  
value 3 value 4  
value 5 value 6
```

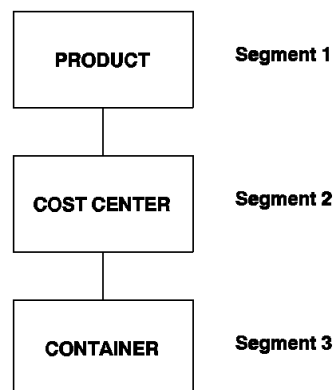
To display data in columns rather than rows (see the following example), set the COLSGLP parameter to Y on the appropriate screen: Update Table SEGLOOP (P170) or Update File SEGLOOP (P175).

```
value 1 value 4  
value 2 value 5  
value 3 value 6
```

Output SEGLOOP Browsing

This topic discusses output SEGLOOP browsing on three segments, with paging.

Database



Data access

Call for Segment 1:

- REQUEST=OUTREAD
- KEY/WHERE=XFER-PRODUCT-KEY (passed from menu)

Call for Segment 2:

- REQUEST=OUTREAD
- Use a Path Call: CMDCODE=D and PROCOPT=P
- OPCODE='>='
- KEY/WHERE=current-segkey-2nd-seg (set with custom code in OCUST2)

Call for Segment 3:

- REQUEST=BROWSE
- KEY/WHERE=XFER-container-key (passed from menu as the start browse key)
- IOAREA=IOA-COSTCENTER-SEGMENT (I/O area of segment 2)

Work areas

Duplicate the PAGE-WORK-AREA to accommodate paging for the second segment. For example:

```
05 PAGES-PAVED-2ND-SEG ...
05 PAGE-KEY-TABLE-2ND-SEG ...
   10 PAGE-KEY-1-2ND-SEG ...
   10 PAGE-KEY-2-END-2ND-SEG ...
05 FILLER REDEFINES PAGE-KEY-TABLE-2ND-SEG
   10 PAGE-KEY-PAVE-2ND-SEG OCCURS 6 TIMES ...
05 CURRENT-SEG-KEY-2ND-SEG ... (SEE NOTE)
```

Note: This should match the KEY/WHERE parameter in the call for segment 2.

SEGLOOP Parameters

The SEGLOOP parameters are as follows:

- PAGEKEY= container-key
- STBRKEY= xfer-container-key
- OCUST2= Custom code member name (sets current-seg-key-2nd-seg)

Custom code

In OCUST2 (keeps key for segment 2 current):

```
MOVE costcenter-key TO current-seg-key-2nd-seg.
```

In OINIT1 (controls paging for segment 2):

```
IF NOT PAGE-FORWARD AND NOT PAGE-BACKWARD
AND PAGE-NUMBER-SAVE = 1
  MOVE 1 TO PAGES-MAVED-2ND-SEG
  MOVE LOW-VALUES TO PAGE-KEY-MAVE-2ND-SEG (1).
IF PAGE-FORWARD
  IF PAGES-MAVED-2ND-SEG = 6
    MOVE PAGE-KEY-2-END-2ND-SEG
    TO KEY-TABLE-2ND-SEG
    MOVE CURRENT-SEG-KEY-2ND-SEG
    TO PAGE-KEY-MAVE-2ND-SEG (6)
  ELSE
    ADD 1 TO PAGES-MAVED GIVING PAGES-MAVED-2ND-SEG
    MOVE CURRENT-SEG-KEY-2ND-SEG
    TO PAGE-KEY-MAVE-2ND-SEG (PAGES-MAVED-2ND-SEG)
  ELSE
    IF PAGE-BACKWARD
    IF PAGES-MAVED = 1
      NEXT SENTENCE
    ELSE
      SUBTRACT 1 FROM PAGES-MAVED
      GIVING PAGES-MAVED-2ND-SEG
      MOVE PAGE-KEY-MAVE-2ND-SEG
      (PAGES-MAVED-2ND-SEG)
      TO CURRENT-SEG-KEY-2ND-SEG (SEE NOTE)
    ELSE
      MOVE PAGE-KEY-MAVE-2ND-SEG (PAGES-MAVED-2ND-SEG)
      TO CURRENT-SEG-KEY-2ND-SEG. (SEE NOTE)
```

Note: This keeps the key for segment 2 current.

PF keys

Incorporate code in the P-100 section to restrict paging forward when no more data is available, or paging backward when a user is already at the beginning.

Consistency Edits in SEGLOOPS

If you want to execute consistency edits for all iterations of a SEGLOOP data item in one pass, CA Telon offers the SEGLOOP switch on the Update XFEDIT (P165) or Update SEGEDIT (P168) screens.

When you set the SEGLOOP parameter to "Y," code is generated to perform the SEGEDIT or XFEDIT for all iterations of the SEGLOOP. If there are any errors, each error is highlighted and displayed in one pass. If you do not set SEGLOOP field to "Y," each iteration is tested one at a time and the error is displayed for only one iteration at a time.

The following is an example of a screen where you can set the SEGLOOP field:

```
TRCC2K.PD UPDATE XFEDIT *****
COMMAND ==> _____
EDIT NAME SEGTST
COPY EDIT BASE: _____ SEGLOOP: _
EDIT CONDITION: _____
_____
_____
_____
ERROR MESSAGE: ERROR MESSAGE HERE _____
HIGHLIGHT FIELDS: _____
ERRCHAR FIELDS: _____
CURSOR AT FIELD: _____
```

Referencing Screen Attributes in PL/I

If you need to reference the screen attribute of a field in your program, it is better to reference the field's TPO_ATTR rather than its TPI_ATTR host variable.

TPI_ATTRs are numbered sequentially as they appear on the screen. When you add a new field to the screen, the TPI_ATTR variables are renumbered.

The TPO_ATTRs are explicitly named using the field name. Therefore, when you add a new field to the screen, you do not have to change your program's custom code.

Individual Field Edit Error Messages

You can issue individual error messages against screen input fields, without using XFEDIT and SEGEDIT X-100 consistency edits, by doing the following:

- Reset CONTROL-INDICATOR at the end of E-100 so that the X-100 consistency edits are always performed
- Generate the desired error message (TPO-ERRMSG1)

At the end of E-100-INPUTS-EDITS (custom code entry point FLDEDIT), code the following:

```
IF NOT CONTINUE-PROCESS
  MOVE CONTINUE-PROCESS-LIT TO CONTROL-INDICATOR
  MOVE 'Y' TO WS-FIELD-ERRORS
ELSE
  MOVE 'N' TO WS-FIELD-ERRORS.
```

In working storage, insert a declaration for the switch used in the code:

```
05 WS-FIELD-ERRORS PIC X.
```

Create consistency edits for all INPUT and OUTIN fields:

```
SRC IF WS-FIELD-ERRORS = 'N'
SRC GO TO TELON-CONSISTENCY-EDITS.
XFEDIT Condition: TOP-field-ATTR = ERROR-ATTR (see Note)
  set desired error message.....
  set attributes & cursor as required.....
XFEDIT .....
XFEDIT etc., one for each screen input field
XFEDIT etc.
SRC TELON-CONSISTENCY-EDITS.
.....
..... normal XFEDIT/SEGEDIT/SRC edits.
```

Note: During E-100 field edit processing, the edit moved ERROR-ATTR to TPO-field-ATTR for each input field in error. Therefore, if TPO-field-ATTR is equal to ERROR-ATTR, that field had an error. Update the XFEDIT to establish the desired error message and screen attributes.

Light Pen Support

This topic outlines the coding procedures for supporting the 3270 light pen feature in CA Telon By inserting custom code into specified locations in the program, you can use the light pen as either an immediate or deferred Enter.

Note: Using the light pen as an actual AID key (i.e., by a short read in IMS or CICS) is *not* possible in CA Telon at this time.

You must address the following areas of the CA Telon screen definition in supporting the 3270 light pen:

- The EOFKEY parameter on the TDF
- The application work area (hhWKAREA)

- The program custom code entry point OINIT1
- The program custom code entry point ININIT1

The following discusses the coding required in each of these areas to support the 3270 light pen feature.

SCREEN/EOFKEY

To use the 3270 light pen feature as either an immediate or deferred Enter key, set the EOFKEY parameter in the Update Screen Params screen to **Y**. This specification results in all input fields having default attributes with the modified tag set (pre-modified).

Application Work Area hhWKAREA

You must add constants that define PEN detectable attributes to the programs. The simplest method adds them into the hhWKAREA copy member, which is copied automatically into every CA Telon online program as the application work area. The attribute settings required are as follows:

For IMS

WS-PEN-ATTR PIC XX VALUE ' '.
Hex value: x'00C2'
Attributes: Unprotected, Normal Intensity,
Pen Detectable

WS-PEN-PROT-ATTR PIC XX VALUE ' '.
Hex value: x'00F2'
Attributes: Protected, Normal Intensity,
Pen Detectable

WS-PEN-HIGH-ATTR PIC XX VALUE ' '.
Hex value: x'00C8'
Attributes: Unprotected, High Intensity,
Pen Detectable

WS-PEN-PROT-HIGH-ATTR PIC XX VALUE ' '.
Hex value: x'00F8'
Attributes: Protected, High Intensity,
Pen Detectable

WS-PEN-CURSOR-ATTR PIC XX VALUE ' '.
Hex value: x'C0C2'
Attributes: Unprotected, Normal Intensity, Cursor,
Pen Detectable

WS-PEN-CURSOR-HIGH-ATTR PIC XX VALUE ' '.
Hex value: x'C0C8'
Attributes: Unprotected, High Intensity, Cursor,
Pen Detectable

Note: The modified data tags in all of the above attributes are preset to the OFF position. This causes each IMS application program to see all pen detectable fields on input as spaces unless the light pen was actually used on the field. This allows you to use the light pen on CA Telon SELECT fields.

For CICS

```
WS-PEN-ATTR PIC XXX VALUE ' '.
Hex value: x'0000C4'
Attributes: Unprotected, Normal Intensity,
Pen Detectable
WS-PEN-PROT-ATTR PIC XXX VALUE ' '.
Hex value: x'0000F4'
Attributes: Protected, Normal Intensity,
Pen Detectable
WS-PEN-HIGH-ATTR PIC XXX VALUE ' '.
Hex value: x'0000C8'
Attributes: Unprotected, High Intensity,
Pen Detectable
WS-PEN-PROT-HIGH-ATTR PIC XXX VALUE ' '.
Hex value: x'0000F8'
Attributes: Protected, High Intensity,
Pen Detectable
WS-PEN-CURSOR-ATTR PIC XXX VALUE ' '.
Hex value: x'FFFFC4'
Attributes: Unprotected, Normal Intensity, Cursor,
Pen Detectable
WS-PEN-CURSOR-HIGH-ATTR PIC XXX VALUE ' '.
Hex value: x'FFFC8'
Attributes: Unprotected, High Intensity, Cursor,
Pen Detectable
```

Note: You must use BMS to support a light pen in the CICS environment.

Program Custom Code Entry Point OINIT1

Use custom code entry point OINIT1 to initialize all pen detectable fields and attributes on output. You must initialize all immediate fields with a literal value beginning with the character **&.**, and deferred-detect fields with a literal value beginning with the character **?**. You must initialize all pen detectable field attributes with one of the above specified attributes.

Program Custom Code Entry Point ININIT1

Use custom code entry point ININIT1 to reset any pen detectable fields and attributes which are, at any point in the program, set to one of the above-defined cursor attributes. All cursor attributes sent to the screen in any iteration are reset to spaces in the CA Telon merge routine on input. This causes MFS to default the attribute on the next output. Since the default MFS attributes are not valid pen attributes, all pen detectable fields must be continually dynamically modified.

Generated Attribute Setting

CA Telon-generated code automatically positions the CURSOR at screen fields for each of the following conditions:

- A field is defined in the CURSOR parameter of the SD screen
- A field edit failed (from FIELD/FLDTYPE specification on the Update Output, Input, Outin Fields screen or the Update Select Field screen)
- An XFEDIT failed (from CURSOR, HILIGHT, or ERRCHAR specification on the Update XFEDIT screen)
- Multiple SELECT-type fields where chosen (when SELECT fields are specified on a panel)
- No SELECT-type fields chosen (when SELECT fields are specified on a panel)

Because the current CURSOR-type attributes are not valid for pen detectable fields, take the following precautions to prevent automatic setting of these fields' attributes:

- Do not specify pen detectable fields in the CURSOR parameter on the Create/Update Screen Definition screen.
- When using SELECT-type fields, specify CURCUS on the Create/Update Screen Definition screen. The CURCUS parameter prevents the cursor from being positioned at the first SELECT field when either no SELECT fields are chosen or multiple SELECT fields are made immediately detectable.
- Do not specify pen detectable fields in the CURSOR, HILIGHT, or ERRCHAR parameters as specified on the Update XFEDIT screen or the Update SEGEDIT screen.

Defining Fields

Define all pen detectable fields as either INPUT or SELECT field types. The first byte of a pen detectable field must contain either a **?** or a **&..** You must move these values into the field using custom code, *not* the INIT parameter in the panel definition for the field. The custom coding requirements are as follows:

C940I

C940I moves spaces to the output buffer for all pen selectable fields (TPO-name). Move the correct pen-detectable attribute to each pen selectable field (TPO-name-ATTR).

```
MOVE WS-PEN-PROT-HIGH-ATTR TO TPO-INADD-ATTR
TPO-INDISP-ATTR
TPO-INUPDT-ATTR
TPO-INLIST-ATTR.
MOVE SPACES TO TPO-INADD
TPO-INDISP
TPO-INUPDT
TPO-INLIST.
```

C940T

C940T moves the values displayed, including the first character of **&.** or **?**, to each pen selectable field.

```
MOVE '&ADD'. TO TPO-INADD.
MOVE '&DISPLAY'. TO TPO-INDISP.
MOVE '&UPDATE'. TO TPO-INUPDT.
MOVE '&LIST'. TO TPO-INLIST.
```

ININIT1

On input processing, only the field selected by the light pen has a value in the input buffer (TPI-name). All other pen selectable fields have spaces in their TPI-name.

```
IF TPI-INADD NOT = SPACES
  MOVE 'X' TO TPI-SELADD.
IF TPI-INDISP NOT = SPACES
  MOVE 'X' TO TPI-SELDISP.
IF TPI-INUPDT NOT = SPACES
  MOVE 'X' TO TPI-SELUPDT.
IF TPI-INLIST NOT = SPACES
  MOVE 'X' TO TPI-SELLIST.
```



```

TRLPEN.PD UPDATE SELECT PARMS ***** SCREEN DEFINITION SAVED
COMMAND ==> PAGE 01
LINE 001 COL 002 SIZE 024 000
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
0001 SAMPLE MENU WITH LIGHT PEN SUPPORT
0002
0003
-----
XFEDIT/
LN COL LTH **NAME** U/S SCONSIS SEGEDIT NEXTPGM INEDIT IMDBIO
04 030 001 SELADD - - - - - ADDS - - - - -
06 030 001 SELDISP - - - - - DISP - - - - -
08 030 001 SELUPDT - - - - - UPDT - - - - -
10 030 001 SELLIST - - - - - LIST - - - - -

```

Minimizing the Size of SPA/COMMAREAs

This topic illustrates how to decrease the size of the SPA/COMMAREA. In a CA Telon generated program, the SPA/COMMAREA consists of three parts:

- Sixteen bytes of control information for program control purposes
- Transfer work area(s) (as long as required)
- Screen image area that holds the screen image for line optimization purposes

Decreasing the SPA/COMMAREA

The 16 bytes of CA Telon control information are mandatory and you cannot modify their length.

The application's transfer work area requirements are the major factor in determining the size of the SPA/COMMAREA. Except for some CA Telon required fields (only required if you use SEGLOOPing or Help or Hold), you control what is included in the transfer work area.

You can drastically reduce the size of the third part of the transfer work area, the screen image area. The screen image area consists of 12 bytes of control information and n bytes of information relating to each field on the screen, where n consists of three bytes + length of field, totaled for each field on the screen, whether INput, Output, output/input (O/I), or SElect fields.

For example, when four fields are on the screen with lengths of 8, 1, 6, and 79, the Screen Image area consists of 118 bytes:

$$12 + (3 + 8) + (3 + 1) + (3 + 6) + (3 + 79) = 118 \text{ bytes}$$

Set REFRESH=**N** (UPdate SD and then update SCRN PARMS). The SCREEN IMAGE no longer sorts OUTput field data and does not save the 3 + 79 for the error message (ERRMSG1) field. This reduces the total to 36 bytes. (See screen handling topics for more details.)

Line Optimization and Screen Merge Processing

One of the parameters that raises many questions from users is the LINEOPT parameter. This subject explains the LINEOPT values.

LINEOPT=1

Specify LINEOPT=**1** to use CA Telon's line optimization subroutines.

LINEOPT=1 for IMS Programs

The following calls are generated:

C-930-INPUT-MERGE

```
CALL 'ADRAMGI' USING      TP-OUTPUT-TABLE
                          TP-OUTPUT-BUFFER
                          TPI-PFKEY
                          SCREEN-IMAGE-AREA.
```

C-940-OUTPUT-MERGE

```
CALL 'ADRAMGO' USING     TP-OUTPUT-TABLE
                          TP-OUTPUT-BUFFER
                          TPI-PFKEY
                          SCREEN-IMAGE-AREA.
```

LINEOPT=1 for CICS Programs

Using BMS, the following calls are generated:

C-930-INPUT-MERGE

```
CALL 'TLRAMRI' USING     TP-OUTPUT-TABLE
                          TP-OUTPUT-BUFFER-FIELDS
                          SCREEN-IMAGE-AREA
                          PFKEY-INDICATOR.
```

C-940-OUTPUT-MERGE

```
CALL 'TLRAMRO' USING      TP-OUTPUT-TABLE
                           TP-OUTPUT-BUFFER
                           SCREEN-IMAGE-AREA
                           PFKEY-INDICATOR
                           EIBAIID.
```

Note: With the non-BMS programs, the merge routine is called from the CA Telon I/O subroutines. These I/O subroutines are called in: C-100-TERMIO-READ and C-200-TERMIO-WRITE.

Merge Fields

The TP-OUTPUT-TABLE contains the information every field needs to perform line optimization. It consists of an 18-byte header followed by an entry for each variable field. The header describes the environment and the options for processing the message (or screen). The variable field entries consist of a halfword field descriptor that defines the field type and a halfword length field. For SEGLOOPS, there is also an entry for the beginning and the end of the SEGLOOP.

The TP-OUTPUT-BUFFER is the buffer that contains the output and input message. For output, the buffer is made up of all variable fields preceded by a two-byte (three bytes for CICS) attribute field. For input, the buffer is redefined for only the input fields with fillers in place of all other variable fields and all attribute fields.

The SCREEN-IMAGE-AREA is part of your SPA/COMMAREA and contains a copy of your screen image.

In IMS programs, the input message is read into TP-INPUT-BUFFER.

TPI-PFKEY in the TP-INPUT-BUFFER marks the beginning of the data portion of the message. In CICS programs, the EIBAIID byte is translated to a numeric value and returned in PFKEY-INDICATOR.

Output Merge Processing

Before writing a message (or screen) to the terminal, the merge routine is called (ADRAMRGE for IMS and TLRAMRG for CICS) to determine what data must be sent. The output buffer fields are compared to the screen image area fields (remember that the screen image area contains a copy of the latest screen image):

- For every field. The attribute from the output buffer is moved to the SCI area.
- If an output buffer field is the same as the SCI (screen image) field (if the field is already on the screen). HEX-3F for IMS or low values for CICS are moved to the output buffer field so this field is not retransmitted to the terminal.
- If the output buffer field is not the same as the SCI field (the value of the field has changed or is not already on the screen). The output buffer field is moved to the SCI field.
- Fields are optionally filled with nulls or underscores, depending on what you specified in OUTIFIL.

Input Merge Processing

Immediately following the reading of the message (or screen) on input, the merge routine is called to update the screen image area.

- A cursor attribute on any field is reset to normal.
- If you entered a field or hit the EOF key, all nulls or trailing underscores are translated to spaces and the field is moved to the SCI area. (It does not translate *embedded* underscores to spaces.)
- If you did not enter a field, the current screen value is copied from the SCI area into the output buffer.
- For CICS, the EIBAID byte is converted to a value and returned in PFKEY-INDICATOR. Also, if you modified at least one field, the SCI-MODIFY-INDICATOR is turned on.

After the input merge, the screen image area and the output buffer (redefined as the input buffer) contain an exact copy of the data on the screen from which you enter the data.

LINEOPT=2

Specify **LINEOPT=2** if you do not want to use CA Telon's called merge routines. This generates all line optimization code within the program, eliminating any calls to CA Telon lineopt routines (for CICS, you must use BMS mapping to use **LINEOPT=2**). In place of the calls in C-930-INPUT-MERGE and C-940-OUTPUT-MERGE, CA Telon generates COBOL or PL/I code that essentially does the same processing as **LINEOPT=1**.

LINEOPT=3

Specifying **LINEOPT=3** generates no line optimization in your program. All data is transmitted on input and output. With **LINEOPT=3**, no screen image area is generated. Therefore, you cannot use HELP and HOLD, since these facilities use the screen image area to hold the screen. Not having a screen image area greatly reduces the size of the SPA (or COMMAREA). For more information, see "Minimizing the Size of SPA/COMMAREAs," earlier in this chapter.

Note: If a 3270 transmission option is being used in the CICS environment, use **LINEOPT=3** to minimize potential conflicts and duplication of function.

CA Telon Screen Handling

Online programs send data to and from the terminal. In a CA Telon program, parameters determine exactly how the data is sent, received, and implemented in the program's code.

Parameter Descriptions

The parameters considered here are **LINEOPT**, **REFRESH**, **ALARM**, **OUTIFIL**, **EOFKEY**, and **OUTATTR**. They are described for CICS programs with and without BMS, and for IMS programs (which always use MFS). Descriptions are given for every allowable combination.

LINEOPT

LINEOPT sends the least possible amount of data to and from the terminal, especially for multiple iterations of a single screen. **LINEOPT=1** optimizes line transmission using called Assembler subroutines. **LINEOPT=2** optimizes line transmission using COBOL or PL/I code-generated online in the program. **LINEOPT=3** does online optimization, thus allowing an installation to call its own optimizing routines.

LINEOPT=1 and 2

For LINEOPT=1 and 2, CA Telon:

- Sends each literal to the screen on the first iteration where it remains for subsequent iterations.
- Sends each output field and respective attribute to the screen on the first iteration. Subsequent iterations resend the field only if it changed. Merge routines handle this process. The attribute is resent every iteration.
- Receives each input field from the terminal only if something was entered in it (except if EOFKEY=Y is set, described later in this subject). The merge routines handle this process.
- Sends the shortest possible representation of each field. Various parameters affect this and are described where appropriate.

Note: OUTIN fields combine OUTPUT and INPUT fields. SELECT fields are INPUT fields.

ALARM=Y,N

ALARM=Y rings the terminal alarm on error iterations (if one or more fields have an error attribute set).

OUTIFIL=SPACES,NULL,UNDERLINE (S, N, U)

OUTIFIL fills the trailing bytes of the input fields sent to a terminal with the specified character. NULL sends only the data and is the most efficient. CICS actually puts nulls in the trailing bytes. In IMS, MFS uses the X'3F' that CA Telon placed there. All input fields are space-filled before the OUTIFIL logic on OUTPUT processing (C-200) and after the OUTIFIL logic on INPUT processing (C-100).

REFRESH=Y,N

REFRESH=Y saves the OUTPUT field values in the transfer work area and keeps them from screen to screen. Thus, if a screen is on the terminal and the operator puts the program on HOLD and then returns, the OUTPUT fields are replaced on the terminal without performing the whole of the A-100 and B-100 again. INPUT fields are always saved. For information on applications that do not require HELP/HOLD, see "Minimizing the Size of SPA/COMMAREAs" earlier in this chapter.

OUTATTR=Y,N for each OUTPUT field

OUTATTR=Y creates a field TPO-fldname-ATTR for the output field. The programmer can modify this field in the code. OUTATTR=N causes the field to have one initial attribute used throughout the program.

EOFKEY=Y,N

EOFKEY=**N** allows only modified INPUT fields to be sent from the terminal to the program. This is most efficient. However, pressing the EOF key in a field does not set that field's Modified Data Tag (MDT) and does not transmit the field back to the program. EOFKEY=**Y** transmits all INPUT fields on every iteration. This allows you to use the EOF key to clear a field and pass that cleared field back to the program.

Allowable combinations

ALARM REFRESH OUTIFIL EOFKEY OUTATTR
LINEOPT=1
 BMS=N Y,N Y,N S,N,U - -
 BMS=Y Y,N Y,N S,N,U - -
 IMS Y,N Y,N S,N,U Y,N Y,N

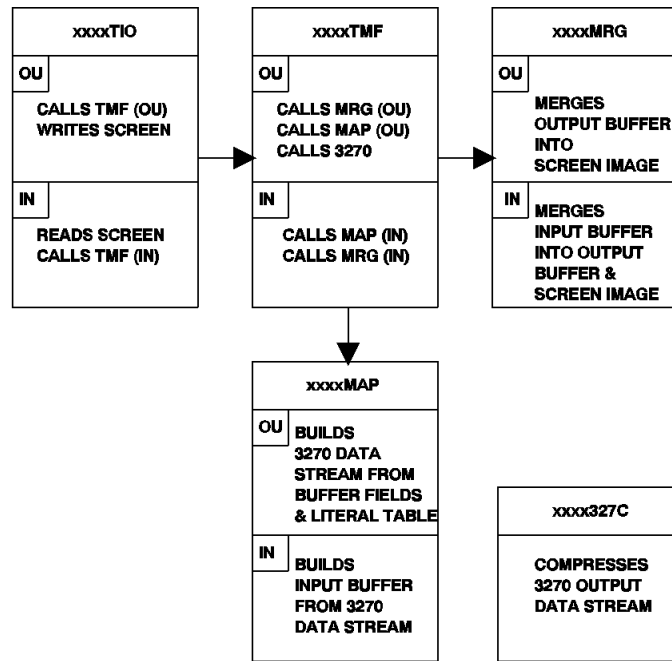
LINEOPT=2
 BMS=Y N Y,N S,N,U - -
 IMS N Y,N S,N,U Y Y,N

LINEOPT=3
 BMS=Y N N N - -
 IMS N N S Y Y,N

Implementation

LINEOPT=1, CICS, BMS=N

Called Assembler subroutines completely handle line transmission, screen merging, and such. Sections C-100 and C-200 call TLRATIO to read from and write to the terminal. See the following diagram for details.



Note: CICS - BMS = YES and IMS programs *ONLY* call xxxxMRG. xxxxTIO = TLRATIO, xxxxTMF = ADRATMF, xxxxMAP = ADTAMAP, xx327C = ADRA327C.TLPATIO is linked to by TLRALIO.

The generator takes the values of ALARM, OUTIFIL, and REFRESH and sets bits in the TP-OUTPUT-TABLE accordingly. It passes this table to TLRATIO, which uses the information to send the screen correctly. The value of the first two bytes of TP-OUTPUT-TABLE reflects whether ALARM equals **Y** or **N**. The value of the first two bytes in the entry for each INPUT field in TP-OUTPUT-TABLE indicates the OUTIFIL choice. In TP-OUTPUT-TABLE, the value of the first two bytes in the entry for each OUTPUT field indicates the REFRESH choice.

Note: If BMS equals N, LINEOPT must equal 1.

LINEOPT=1, CICS, BMS=Y

The Assembler routine handles the merge, and BMS performs the rest of the transmission. The TP-OUTPUT-TABLE has flags to indicate whether ALARM and REFRESH are **Y** or **N**, and to indicate the OUTIFIL value. The merge routine handles the REFRESH and OUTIFIL logic. For ALARM, the routine determines if this is an error iteration. If so, a flag in the screen image area is set on. This flag is interrogated in C-210 to decide if the BMS SEND MAP should use the BMS ALARM parameter.

LINEOPT=2, CICS, BMS=Y

The merge routine is generated in native COBOL or PL/I code in C-930 and C-940. REFRESH=Y includes code for the OUTPUT fields in the merge routine. For OUTIFIL=SPACES, no additional code is necessary as the fields have trailing spaces by default. For OUTIFIL=NULLS and UNDERLINE, **C-948** is performed for every INPUT field to replace their trailing spaces on output, and to replace the trailing nulls or underscores with spaces on input.

LINEOPT=2, IMS

The merge routine and REFRESH and OUTIFIL are handled as in BMS. OUTIFIL=NULL fills the INPUT fields' trailing bytes with X'3F' instead of binary zeros because that is what MFS uses. Upon input, the X'3F' in INPUT fields is replaced with SPACES. EOFKEY must equal Y and is handled like LINEOPT=1. For OUTATTR=Y, the OUTPUT field attributes are handled in the routines along with the other attributes.

LINEOPT=3, CICS, BMS=Y

No line optimization is performed except to use the BMS DATAONLY parameter. This parameter sends only changed fields to the terminal on the second and subsequent screen iterations. Whatever you move into fields in output processing is always sent. OUTIFIL must equal NULL so INPUT fields that you partially enter are NULL-filled; fields that you did not enter are filled with spaces in C-990.

LINEOPT=3, IMS

OUTATTR=Y generates attribute fields for each OUTPUT field. You can modify these in the program.

Using the Configuration Section

If you want to use the configuration section in a generated COBOL program (to define special-names, for example), TLNIIS provides a parameter to copy in a user-defined member. To use the configuration section, you must follow these steps:

- Use ISPF select to update the correct TLNIIS member on pdsqual.MACLIB
- Locate the SETSYS statement and change the CONFCPY parameter to CONFCPY=CONFCPY.

- Add a member called CONFPCY to the pdsqual.SOURCEC data set. This member contains the required default configuration section code. It might contain just one line of comment. For example:

```
* THIS IS A DUMMY CONFIGURATION SECTION MEMBER
```

- Create a custom code member called CONFPCY for the required program within the TDF when a program requires anything other than the default. This is automatically copied in at compile time. An example of the member is:

```
SPECIAL-NAMES.  
DECIMAL-POINT IS COMMA.
```

3270 Numeric Lock Feature

The numeric lock feature allows a 3270 terminal to detect non-numeric input. All 3270 type terminals do not necessarily have this feature.

To use this feature with CA Telon, you must add two new attributes. You can code these attributes anywhere in working storage in the hhWKAREA. The definitions are as follows:

```
* NUM-OK-ATTR X'0000D0'  
05 NUM-OK-ATTR PIC X(3) VALUE ' ' .  
* NUM-ERROR-ATTR X'FFFD8'  
05 NUM-ERROR-ATTR PIC X(3) VALUE ' ' .
```

Note: You must enter the values in HEX in the 05 level definitions. Use TSO/ISPF and invoke the HEX command to enter these values.

For TSO/IMS, the attribute byte lengths are two characters; CICS extended attributes use three characters. These values are 00D0 and FFD8 for TSO/IMS.

You can then use logic in the custom code entry points to move the defined attributes to the TPO-field-ATTR variables.

CA Telon Help Facility

CA Telon can generate programs with their own built-in CA Telon help logic. These programs have two levels of help: screen- and field-level help.

CA Telon Screen-Level Help

Use custom code to implement CA Telon Screen-Level Help, and PF-key logic to invoke it. In look and feel, screen-level help is similar to the IBM ISPF help facility. Screen-level help displays one or more pages of help information that discusses the current panel image. Enter (or copy) a few lines of source code into PF-key custom code entry points to implement screen-level help.

CA Telon Field-Level Help

CA Telon Field-Level Help provides you with a more detailed level of help about each input field on the panel image. You simply enter a question mark (or any other installation definable character) into the field(s) you need help with, and press Enter. The Help Display Program then presents a screen of information for each of the fields selected. Place a **Y** next to the HELP/HOLD parameters in the Update Screen Parm's panel to implement field-level help. Also, you can enter field-level help message keys (HELPMMSG) against each input field in the update HELPMMSG parameter panel accessed from the Update Panel Definition screen.

When you invoke either of these help levels, CA Telon either generates or builds, with PF-key logic, a small table of help key values in the transfer work area. CA Telon also saves the status of the current program (performs HOLD processing) and transfers control to a Help Display Program. The Help Display Program uses the help keys passed in the transfer work area to access a help data set and displays the selected help pages. When you exit from the Help Display Program, control passes back to the original calling program. This program restores itself to its original state (resume from HOLD) and continues processing. The transfer work area required to support CA Telon HELP and to pass the help data set keys to the help display program follows.

```

EDIT —— TRCTDA.SD TRXFERWK MEMBER 001 OF 001 SIZE 000075 COL 87
COMMAND ==> SCROLL ==> CSR
000024 **-----**
000025 ** FIELDS REQUIRED BY TELON GENERATED APPLICATIONS USING **
000026 ** HELP AND/OR HOLD PROCESSING **
000027 **-----**
000028      05 XFER-HELP-AREA.
000029          10 XFER-HOLD-INDICATOR   PIC X.
000030          10 HELP-CURR-MSG-COUNT    PIC 99 COMP-3.
000031          10 HELP-MSG-COUNT         PIC 99 COMP-3.
000032          10 HELP-MSG-NAME          PIC X(8)
000033                                     OCCURS 10
000034 TIMES.
000034 **-----**

```

The following custom code in member hhPFK1 supports screen-level help for a program. You can copy this example code directly from pdsqual.SOURCEx into a program using the PDSCOPY editor command. You can then make any necessary modifications.

```

EDIT —— TRCTDA.SD TRPFK1 MEMBER 001 OF 001 SIZE 000018 COL 87
COMMAND ==> SCROLL ==> CSR
***** ***** TOP OF DATA *****
000001 *****
000002 *
      *
000003 *
      *
000004 * FUNCTION KEY 1/13 IS USED TO REQUEST SCREEN LEVEL HELP *
000005 *
      *
000006 *
      *
000007 *****
000008         IF PFK1-13
000009             MOVE 'P' TO HOLD-AREA-TYPE
000010             PERFORM L-100-HOLD-SAVE
000011             MOVE PROGRAM-NAME TO HELP-MSG-NAME(1)
000012             MOVE 1 TO HELP-MSG-COUNT
000013             MOVE 1 TO HELP-CURR-MSG-COUNT
000014             MOVE HELP-SCREEN-PGM-ID TO NEXT-PROGRAM-NAME-ID
000015             MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
000016             GO TO P-100-PFKEYS-RETURN
000017         ELSE
000018             NEXT SENTENCE.
***** ***** BOTTOM OF DATA *****

```

You can implement multiple help screens for screen-level help for a program simply by inserting additional MOVE statements. These statements move other keys to the HELP-MSG-NAME array after statement 11 in the above example code and set HELP-MSG-COUNT to the number of keys now in the array. For example:

```

MOVE 'TRCTDA2' TO HELP-MSG-NAME(2)
MOVE 'TRCTDA3' TO HELP-MSG-NAME(3)
MOVE 3 TO HELP-MSG-COUNT

```

For information on HELP/HOLD Processing, see Chapter 9, "Processing Flow."

Note: Set ERROR-REQ-CHAR (usually *) in member hhWKAREA to the same value as HELP-CHAR (usually ?). When you do not enter a required field or fields, question marks appear in all mandatory fields after you press Enter. You can then enter all the *known* fields and press Enter again to get field level help for all the *unknown* fields. This provides you with a very user-friendly final product.

Appendix A: Using the IMS Transaction Manager Environment

This appendix specifically addresses the IBM IMS Transaction Manager environment.

For information on topics relating to the IMS/DB and DL/I Database Management Systems, see Appendix E, "DL/I Specific."

This section contains the following topics:

[Exiting IMS Application Programs](#) (see page 561)

[WORKSPA Database](#) (see page 562)

[Combined SPA/WORKSPA Database Handling](#) (see page 565)

[Dynamic Link Programs](#) (see page 566)

[Static Link Program](#) (see page 570)

[HELP/HOLD Facilities](#) (see page 577)

[Writing Non-Screen Transactions](#) (see page 581)

[IMS/DC Report Processing](#) (see page 582)

[CA Telon Driver Applications](#) (see page 587)

[CA Telon Program Transfers](#) (see page 587)

[Program Switching to Non-CA Telon Programs Using MFS](#) (see page 591)

[Using Multilingual MFS Screens](#) (see page 592)

Exiting IMS Application Programs

To exit IMS applications normally, put the following code in the output side of your program (usually) through the OINIT1 custom code entry point:

```
MOVE SPACES TO SPA-TRANSACTION-CODE. (COBOL)
```

- or -

```
SPA-TRANSACTION-CODE = ' '; (PL/I)
```

For conversational programs, this releases the IMS Core SPA. For non-conversational programs, this deletes the WORKSPA database.

After an interruption occurs (i.e., IMS goes down and your program is using WORKSPA or HOLD databases) and you re-enter the system:

- The WORKSPA database is re-initialized to low values
- You can delete the HOLD database, if present, by invoking the return from the HOLD PF-key code

WORKSPA Database

Note: For further information, read this topic in conjunction with similar topics in the *Implementation* manual.

CA Telon-generated applications use a transfer work area to pass information between programs. For the IMS environment, this area is contained on a WORKSPA database or IMS Core SPA.

A conversational program uses the IMS Core SPA. If all transfer work area requirements do not fit into your IMS Core SPA, it can use a WORKSPA database. The overflow goes onto the WORKSPA database.

Creating the WORKSPA Database

The WORKSPA database is generally a simple HDAM database, keyed on LTERM-ID. Your IMS SYSGEN must make it known to IMS.

You can use one WORKSPA database per application, or one for the entire installation. In the latter case, calculate the length of the root segment for the application with the smallest transfer work area requirements using the formula given below.

The lengths of the children reflect the differences in size requirements between applications. For example:

- Application A: total length needed $\geq$ 1000 bytes
- Application B: total length needed $\geq$ 1500 bytes
- Application C: total length needed $\geq$ 2200 bytes

1000 bytes	Segment 1
500 bytes	Segment 2
700 bytes	Segment 3

Application A uses the root only, Application B uses the root and second-level child, and Application C uses all three segments.

To create a WORKSPA database:

- Ensure HDAM, key = item-id, keylength = 8 bytes
 - Ensure Segment length = 22 bytes overhead
 - + 8 bytes key
 - + transfer workarea size
 - + largest screen image size*
- * This is given in the program summary data output from the //GEN step of your generate, compile, link JCL (JIPGCL or JIXGCL)
- Do your DBDGEN
 - Make it known to IMS (IMS SYSGEN)
 - Include the WORKSPA database in your PSBs

If you are not sure of the length of your transfer work area(s), the easiest initial approach is to set up one WORKSPA database (used by everyone), root only, segment length = 2000. This should, in general, meet everyone's needs.

It is suggested that all work relating to establishing standards for the use of the WORKSPA database be done as early as possible in the initial use of the product.

Defining a DB2 Table as a WORKSPA

The following information describes how to define a DB2 table as a WORKSPA:

```
*****
* DCLGEN TABLE(TELON.WORKSPA) *
* LIBRARY(CSAP.REL20B.DCLLIB(WORKSPA)) *
* QUOTE *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS *
*****
EXEC SQL DECLARE TELON.WORKSPA TABLE
( WORKSPA_USERID CHAR(8) NOT NULL,
  WORKSPA_LEVEL SMALLINT NOT NULL,
  WORKSPA_DATA VARCHAR(4000) NOT NULL
) END-EXEC.
*****
* COBOL DECLARATION FOR TABLE TELON.WORKSPA *
*****
01 DCLWORKSPA.
10 WORKSPA-USERID PIC X(8).
10 WORKSPA-LEVEL PIC S9(4) USAGE COMP.
10 WORKSPA-DATA.
49 WORKSPA-DATA-LEN PIC S9(4) USAGE COMP.
49 WORKSPA-DATA-TEXT PIC X(4000).
*****
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 3 *
*****
```

Using the WORKSPA Database

CA Telon controls all I/O involving the WORKSPA database through high-level design parameters.

A WORKSPA request is inserted in TDF Option 4 (update the Data Group) and under each required WORKSPA segment.

- **REQUEST = WORKSPA** (for each segment required)
- **KEY/WHERE = IOPCB-LTERM** (against the root segment)
- **PROCOPT = AP** (if accessing more than one segment)

Note: If you want to create a static link program, the driver handles all WORKSPA database processing. Thus, you should perform the above steps for the driver program only, not the individual screen modules under that driver.

Combined SPA/WORKSPA Database Handling

The Spa area generated in the COBOL program equals the combined length of the IMS Core SPA and the WORKSPA, less the eight-byte WORKSPA key. This corresponds to either a 14 or 22 byte header, plus the program's transfer work area (XFERWKAs).

In the C-950-PUT-WORKSPA section of an IMS online program:

- The first eight bytes of the WORKSPA portion of the SPA-AREA (called WORKSPA-KEY) are saved in SPA-WORKSPA-DHOLD.
- The WORKSPA portion of SPA-AREA then has the eight-byte WORKSPA-KEY-FB-AREA moved into its first eight bytes. This is the key to the root segment of the WORKSPA database.
- The WORKSPA portion of SPA-AREA is REPL'd to the WORKSPA database.
- The eight bytes saved in SPA-WORKSPA-DHOLD are restored to the first eight bytes of the WORKSPA portion of SPA-AREA.

Next, in the C-960-PUT-SPA section, the IMS Core SPA itself is written. Using the IO-PCB, the program attempts to insert the full SPA-AREA (combined length of IMS Core SPA and WORKSPA minus eight). However, the TRANSACT macro defines the IMS Core SPA length as shorter than SPA-AREA. Only that much of it is written, including as its last eight bytes the field WORKSPA-KEY. This field is actually only eight bytes of data, not a key at all.

Example

If for instance:

SPASIZE = 2000, WKSPASZ = 2000

Then the SPA-AREA consists of:

```
FILLER PIC X(1992) .  
WORKSPA-AREA PIC X(2000) .  
WORKSPA-KEY REDEFINES WORKSPA-AREA PIC X(8)
```

C-950-PUT-WORKSPA section

- The first eight bytes of the WORKSPA-AREA is saved into SPA-WORKSPA-DHOLD.
- The WORKSPA-KEY-FB-AREA is moved into the first eight bytes of WORKSPA-AREA (i.e., the actual key to the WORKSPA database, LTERM-ID). The WORKSPA-KEY-FB-AREA (not WORKSPA-KEY) is written together with the rest of the WORKSPA-AREA (2000 bytes in all) to the WORKSPA database.
- The saved eight bytes of data from SPA-WORKSPA-DHOLD is restored to WORK-SPA-KEY. This is not really a key, just eight bytes of information from the transfer work area.

C-960-PUT-SPA section

- The FILLER of 1992 bytes plus the first eight bytes of WORKSPA-AREA (i.e., WORKSPA-KEY) are inserted into the IMS Core SPA, a total of 2000 bytes of data.
- The IMS Core SPA is retrieved, followed by the WORKSPA, into the SPA-AREA again ensuring that no data is overwritten. In C-900-GET- WORKSPA, the SPA is stored before the read of the WORKSPA database, and restored to the SPA-AREA after the read of the WORKSPA database.

Dynamic Link Programs

This topic helps you transfer a CA Telon program to the IMS environment as a dynamically linked program. Read this topic in conjunction with similar topics in the *Implementation* manual.

With dynamic linking, each program is a stand-alone unit with its own transaction code, PSB and MFS. Dynamic Programs' IMS messages switch between each other. They can be conversational or non-conversational. After you test a dynamic link program, you must perform the following to move a CA Telon program into the IMS environment ready for acceptance testing, and so on.

IMS SPA/WORKSPA Database

A conversational program uses the IMS Core SPA. It optionally uses a WORKSPA database if all transfer work area requirements do not entirely fit into the IMS Core SPA. Non-conversational programs use a WORKSPA database. For more details about the database, see the earlier subject, [WORKSPA Database](#). (see page 562)

Creating a PSB

Either create an IMS-oriented PSB or let CA Telon do it for you. To create your own PSB, you need:

- An alternative TPPCB for message-switching:

PCBNAME = **XFER**

EXPRESS = **N**

- An alternative TPPCB for the ABEND routine:

PCBNAME = anything you like

EXPRESS = **Y**

ABCALL = **Y** (do this in Data Administration, TDF Option 2, after you import the PSB)

- PCBs for your application databases
- A WORKSPA database PCB:

PCBNAME = **WORKSPA**

PROCOPT = **A** or **AP**

TYPE = **DB**

Include the above in hhPROC and hhPCBS (see TRPCBS and TRPROC in pdsqual.SOURCE or pdsqual.SOURCEP for examples). You must include your IOPCB in hhPROC and hhPCBS, even though it is not explicitly coded in your PSB.

If CA Telon creates your PSB source code, CA Telon creates it according to the PROCOPTs as they exist on the individual update screens for each segment of each database you use.

TDF

- If the production program does not use HELP and HOLD, make sure they are turned off (Update ScreenParms and set HELP=**N**, HOLD=**N**).
- In Option 4, Update Data Group for the WORKSPA database.
Refer to the WORKSPA topics earlier in this appendix.

- In Option 4, CReate ENvironment -IMS:
 - Set LINKOPT=**D** (for Dynamic link).
 - Set CONVERS=**Y** or **N**. **Y** if conversational; otherwise, **N**.
 - Set SPASIZE=**nnnn**. If conversational, enter your installation Core SPA size, as in the IMS SYSGEN; otherwise, leave blank.
 - Set WKSPASZ=**nnnn**. Enter the combined lengths of your WORKSPA database segment(s). If non-conversational, enter the length of your WORKSPA database segment(s). If conversational, use only if your transfer requirements do not fit into the IMS Core SPA (this overflow goes onto the WORKSPA database).
 - Set GENPCBS=**Y** or **N**. Specify **N** to use your own PSB. CA Telon copies your hhPROC and hhPCBS copy books or include members. Specify **Y** if you want CA Telon to generate the PSB.
 - Set LINEOPT=**1**, **2**, or **3**. Enter **1** if you want CA Telon to automatically perform line optimization for you. Enter **2** if you want to simulate subroutine optimization in custom code. Enter **3** if you do not want line optimization code. If you enter 3, you cannot use HELP, HOLD, screen refresh, or underscore in OUTIFIL.
 - Set VALID PSB TRANCOD= trancode desired. Overrides the default CA Telon generated one.
 - Set TRANFLD TRANMFS=**Y** or **N**. **Y** puts the trancode in MFS. Required for non-conversational programs.
 - Set LINKPGM MSGPGM, MSGTRAN, or MSGTABLE; these three fields are mutually exclusive. They contain a list of all screen IDs and programs to which this program can transfer control. Use MSGPGM when using CA Telon-generated transaction codes; otherwise, use MSGTRAN. Only use MSGTABLE if control can pass to many different programs. This field saves the developer from entering all the transaction codes in every CA Telon program.
 - Set SPACMPT=**Y** or **N**. **Y** allows message switching to or from any static program.
 - Set PGMNAME= load module name desired (same as trancode) if not using the default CA Telon-generated names.
 - Set MFSMOD=**MODname**, if different from the generated MODname. Used only to override the default and create a meaningful name for users when the /FORMAT command is used.
 - Set SYSMSG= name of an output field on the MFS screen to which IMS system messages are sent (usually ERRMSG1).

Initializing Fields in the Transfer Work Area

For the first program in a chain, you might want to initialize some fields in the transfer work area. You can do this in custom code entry points, after CA Telon establishes it for you. If you are:

- Not using /FORMAT to invoke the transaction – Initialize any necessary transfer work area fields in OINIT1.
- Always using /FORMAT – Do this in ININIT.
- Sometimes using /FORMAT – Do it in both OINIT1 and ININIT. Take care not to override any values put there during output processing.

For the last program in a chain, if conversational:

- Move spaces to SPA-TRANSACTION-CODE in OINIT1 or OINIT2 (see [Exiting IMS Application Programs](#) (see page 561)).
- Send a message back to the user (using ERRMSG1) stating that the conversation has terminated.

JCL

If you use either JIXGCL or JIPGCL, set the following symbolic parameters:

TDFMEM	hhnnnn	(+SD if using JIPGCL)
DEFTYPE*	SD	
OPTION	0 or 1	0 if CA Telon creates PSB and MFS source code 1 if CA Telon creates only MFS source code
CPARM		as desired
LPARM		as desired
ENV*	I	IMS
PSB*	I or N	I if CA Telon creates PSB source code, otherwise N
FORMAT*	M	create MFS source code
PSBSRC	xxxxxxx	PSB source code library
PSBMEM	xxxxxxx	desired member name of your PSB source
MFSSRC	xxxxxxx	MFS source code library
MFSMEM	xxxxxxx	desired member name of your MFS

source

* If you are using JIPGCL, you should enter these parameters on your utilities screen during your online export instead of in the JCL.

Other Considerations

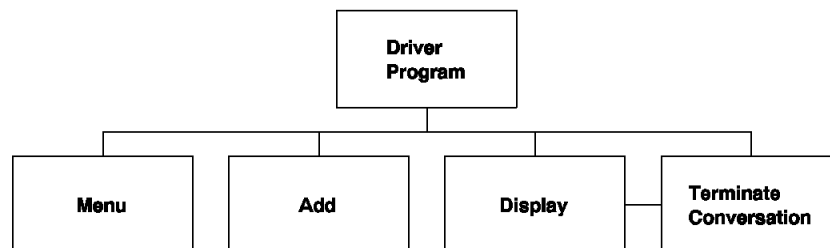
- You must generate the MFS and PSB source code
- When not using CA Telon-generated names, take care to keep the program name, PSB name, trancode, and load module name in synchronization
- You cannot IMS SYSGEN CA Telon programs as enquiries since they all do inserts, deletes, and replaces to the WORKSPA database

Static Link Program

Note: Read this subject with similar topics in the *Implementation* manual.

The CA Telon static link concept packages several screen-oriented programs together under the control of a driver. The driver does not perform screen processing at all; it simply controls the processing done by the various programs or modules under it and their relationship to each other.

A typical structure can be:



Each screen definition is compiled, link-edited, and tested alone, and then included with the driver as one large load module (see [Linking](#) (see page 576) later in this subject for details).

There is one IMS trancode and one PSB associated with the driver. Each screen module has its own MFS.

The program starts from an IMS terminal when you enter either the transaction code or /FORMAT. If you enter a transaction code, IMS schedules the driver that first performs a GU to retrieve the SPA (if conversational) and then a GN to read the message. The driver program calls the first program (typically a menu) for output processing.

If you enter **/FORMAT**, IMS displays the relevant screen image (for example, MOD) and you enter the required data. Processing then continues as if you entered a transaction code, except that the first program is called for input processing.

If errors are encountered, the module's generated logic sets CONTROL-INDICATOR to 'DO-WRITE', sets LINKAGE-WRITE-XFER-INDIC to **W** and returns control to the driver. The driver inserts the SPA (if conversational) and a message to the IOPCB (i.e., send the screen with an error message back to the user).

If no errors are encountered, the screen module completes its own processing, and then tells the driver what to do next using LINKAGE-WRITE-XFER-INDIC:

- X = call the next screen module, according to the name in the NEXT-PROGRAM-NAME-INDICATOR (see program section C-300-TERMIO-XFER).
- W = write the SPA and output message to the screen (IOPCB); the conversation is not yet terminated (see program section C-200-TERMIO-WRITE).

Beginning and Ending a Conversation

When you enter the program for the first time (e.g., from an IMS sign-on screen), the driver requires low values in SPA-NEXT-PROGRAM-NAME to signal the beginning of a conversation. Normally, this occurs automatically. However, if your IMS sign-on screen changes this value, you can reset it in the custom code opening INIT of your driver.

To terminate a conversation, the driver requires spaces in SPA-TRANSACTION-CODE. The recommended method calls a screen module whose only task is to do this. This exit would output an *end of conversation* message for the user (in the form of screen literals), move spaces to SPA-TRANSACTION-CODE (in custom code entry point OINIT1), and return control to the driver (this happens automatically). See the CA Telon training system for an example of this style of program.

Preparing the Modules for IMS

Using the CA Telon Design Facility, update the modules in the static link package.

Driver

Create a driver program, TDF Option **OS (CR DR)**, and update the parameters as described below :

Parameter	Value	Description
CUSTOM CODE		Names of any custom code members slotted into the driver. The only required member is XFERWKA (the transfer work area(s) used by all programs in the package).
FRSTPGM	nnnn	Screen ID of first screen module called (typically, a menu screen).
HOLD	Y or N	Is HOLD being used?
LANGLVL	nnnn	Defaults to latest version of CA Telon (e.g. 4.1)
LANG	COB or PLI	Defaults to COB (COBOL)
REMARKS		Names of any custom code members slotted into the remarks section of the driver. You should mention the names of the screen modules linked to this driver here.

Create an IMS environment for your driver and update the parameters as described below :

Parameter	Value	Descriptions
CONVERS	Y or N	Conversational or Non-conversational (turn off TSO-oriented traces).
FRSTMOD	xxxxxxx	Use only if the MOD name of your first NAME screen module is not CA Telon generated (i.e., if you used MFSSMOD in your first screen module's environment). It must be the same name you used for MFSSMOD LINKPGM(IDs), a list containing all the called sub-modules.

Parameter	Value	Descriptions
GENPCBS	Y or N	Y indicates that CA Telon should generate the PCBs for your program. N causes CA Telon to generate COPY/%INCLUDE statements for your PCB copybook (hhPCBS) and USGCOPY (hhPROC).
IOASIZE	nnnn	The length of the longest segment being accessed in any of the screen modules. The length given here must not be too small; however, too large does not hurt.
MSGPGM (ID's)	nunn	Screen IDs of all independent programs to (ID's) which this driver can message switch. Use only if the independent programs have CA Telon- generated names. Alternatively, you can code ANY.
MSGTBL	XXXXXXXX	An alternative to MSGTRAN if the list is too long.
MSGTRAN (ID, TRAN)	nunn	Screen IDs, followed by IMS trancode, of (ID, TRAN) all independent programs to which this driver can message switch. Use only if the independent programs do NOT have CA Telon- generated names. If this list is too long, use MSGTABLE instead.
PSB/PGMNAME	xxxxxxx	Your load module name, if not using CA Telon- generated names. This must be the same as the IMS trancode.
SPACMPT	blank	Do <i>not</i> use this parameter for a static link driver. size, you must also use the WORKSPA database.
SPASIZE	nnnn	Your installation SPA size, as in your IMS SIZE SYSGEN. If not large enough to hold yor transfer work area + overhead + largest screen size, you must also use the WORKSPA database.
TPISIZE	nnnn	As above, except for your input. Look for DIBUFSZ.
TPOSIZE	nnnn	Maximum size of your TPO (output) buffer, as set in your CA Telon installation. You can browse CA Telon MACLIB (member TELONIIS), to determine this size; look for DOBUFSZ.

Parameter	Value	Descriptions
TRANCDE	xxxxxxx	Use only if your transfer requirements do not fit into the IMS Core SPA (the overflow goes onto your WORKSPA database). Enter the segment length of your WORKSPA database.
WKSPASZ	nnnn	Use only if your transfer requirements do not fit into the IMS Core SPA (the overflow goes onto your WORKSPA database). Enter the segment length of your WORKSPA database.

Create a data group for your driver:

- The data group for your driver must be the same as for all screen modules attached to this driver. Any differences cause the parameter list being passed between the driver and the modules to be out of sync.
- If you use the WORKSPA database in addition to the IMS SPA, the driver has complete control of both. Thus, the driver's DG screen should perform I/O for the WORKSPA database, not the individual screen modules. See the prior topic, [WORKSPA Database](#) (see page 562).
- To create PSB source code automatically, CA Telon generates default PROCOPTs depending on the USAGE of each segment. To override the default values, specify a PROCOPT at either the database or segment level.

Note: When using the WORKSPA database, the PROCOPT must be **A** or **AP**.

Screen Modules

Clean-compile and test each screen module. Then create an environment screen for each screen module linked to the driver and update the parameters as described below :

Parameter	Value	Descriptions
LINKOPT	S	Static link.
CONVERS	Y or N	Conversational or Non-conversational.
SPASIZE*	blank	Leave blank
WKSPASZ*	blank	Leave blank

Parameter	Value	Descriptions
GENPCBS	Y or N	Y indicates that CA Telon should generate the PCBs for your program. N causes CA Telon to generate COPY/%INCLUDE statements for your PCB copybook (hhPCBS) and USGCOPY (hhPROC).
TRACE	N	Turn off TSO-oriented traces after fully testing individual screen programs.
TRANCDE	xxxxxxx	Same as Driver's, or blank if driver's is blank. Must leave blank. A /FORMAT command cannot directly enter a screen module controlled by a driver.
TRANMFS	blank	
TRANFLD	blank	Must leave blank.
LINKPGM*	blank	Must leave blank.
MSGPGM*	blank	Must leave blank.
MSGTRAN*	blank	Must leave blank.
MSGTBL	blank	Must leave blank.
MFSMOD	xxxxxxx	Creates a MOD name other than the CA Telon generated one. If used for the first screen module, this name must be the same as on the driver's environment screen FIRST MOD NAME parameter.
SYSMSG	xxxxxxx	Specify the field where IMS system messages are sent. Typically ERRMSG1.
MIDONLY		Creates, deletes, and updates MID-only fields.
PGMNAME	xxxxxxx	Enter the load module name of this screen module if not using CA Telon-generated names.
SPACMPT	blank	Leave blank. Do not use this parameter for a static link module.
WKSPAIO	blank	Leave blank. The driver controls all WORKSPA database I/O.
LINEOPT	n	1) Use a CA Telon-supplied line optimization routine through a called module. 2) Use CA Telon-supplied line optimization. The code generates in-line in the program. 3) No line optimization. This is not permitted if using the HELP or HOLD facility or OUTIFIL=U.
*These parameters are being controlled in the driver.		

Linking

After creating the environment screens for each screen module, you must individually compile and link-edit them. You can also create the MFS source code at this stage. Copy the compile JCL (JIXGCL) and set the following symbolic parameters:

TDFMEM	hhnnnnn
DEFTYPE	SD
OPTION	1 if CA Telon creates the MFS source, otherwise 3
CPARM	as desired
LPARM	NCAL plus whatever else desired
ENV	I
PSB	N
FORMAT	M for MFS code generation otherwise N
MFSSRC	MFS source code library
MFSMEM	desired member name of your MFS source code

After running, check that your macro listing contains IMSPGM and IMSMFS macros. If it does not contain IMSPGM and IMSMFS, review your JCL parameters.

Compile and link-edit the driver using a copy of the compile JCL. Include the load modules of all the screen modules in the link-edit step. Set the following symbolic parameters:

TDFMEM	hhnnnnn
DEFTYPE	DR
OPTION	2 if CA Telon creates the PSB source code, otherwise 3
CPARM	as desired
LPARM	as desired (NOT NCAL)
ENV	I
FORMAT	N
PSB	I for an IMS PSB or N if no PSB source is required
PSBSRC	PSB source code library
PSBMEM	desired member name of your PSB source code.

Your macro list should contain IMSDRV and IMSPSB macros.

Because the linkage editor automatically tries to resolve all external entries, the load modules of the individual screen modules are included as long as you use your installation's standard library conventions.

Note: Generate the MFS and PSB source code. Verify that HELP and HOLD are turned off everywhere they are not used.

If you are not using the CA Telon-generated names, take care to keep the program name, tranocode, and load module name in sync.

HELP/HOLD Facilities

The following text describes how HELP functions in the IMS environment:

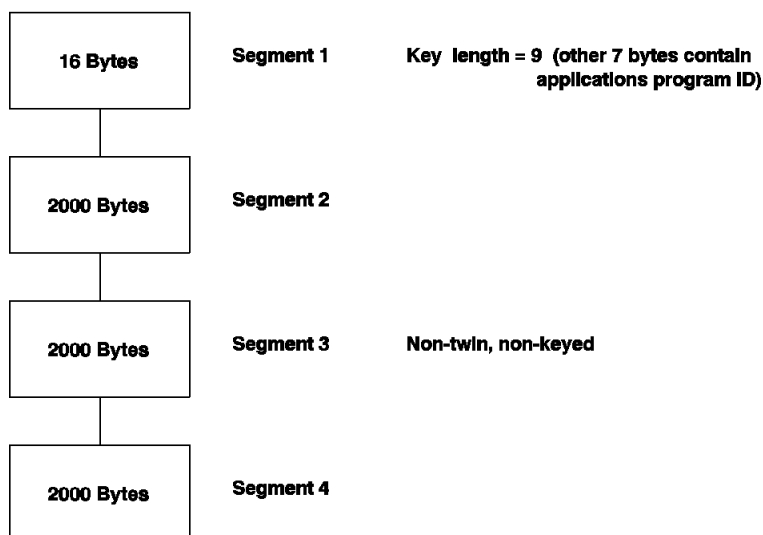
- Enter a **?** into one or more fields for which you require help, or press a predetermined PF key for screen-level help.
- An ISRT to the hold database takes place, inserting the key, transfer work area, and screen image. The key can be either the item-id or the user ID. If the key is a user ID, the user can issue a hold at one terminal and resume from another.
- The application program then transfers control to the help display program defined in hhWKAREA, passing it any help key(s). The help key is the current program's ID if you requested screen-level help, or the value entered into a field's HELPMMSG parameter if you requested field-level help.
- The help program in section A-100 does a GU on the help database. If the status code is GE, it sends a message (by TPO-ERRMSG1) stating that no help is available for that program or field. If a help segment is found (status code is blanks), the help message displays.
- In either case, press PF3 to return to the application program. The help program then does a GU on the root of the hold database to obtain the name of the application program to return to, and the message switches back.
- On return, the application program in section **K-100** does a GHU on the hold database (all segments, path call) to retrieve the stored information. It then deletes those segments on the hold database and continues processing.

CA Telon automatically handles all the above calls. See the TRCTSH or TRPTSH programs in the training system for an example of a help display program.

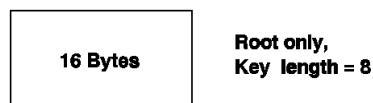
HELP/HOLD Databases

Two online databases are needed, as in the following examples:

- A HOLD database (HDAM) with **PROCOPT=AP** saves the status of the program requesting Help display services:



- A HELP database (HDAM) with **PROCOPT=A**, used by the Help display program to retrieve held screen pages



In both cases, you can use any names, segment sizes, and levels desired for the databases and segments, provided they are adequate. The supplied sample programs use both HLDDBDV1 and HLPDBDV1.

Working Storage fields are needed. For more information, see the Chapter 9, "Required HELP/HOLD Fields".

The fields needed in the transfer work area allow users to enter more than one ? at a time and pass the associated HELPMMSG key(s) to the Help display program.

See TRPFK1 in pdsqual.SOURCEC (COBOL) or pdsqual.SOURCEP (PL/I) for an example of screen-level processing logic.

Setting Up TDF Help Panels

In each application program with help available to its end users, you must indicate the help database key of each field for which help is available.

- In Option 3 (UP PD), use the list HELPMMSGs option (or go into the update screen behind each appropriate field). Fill in the parameter HELPMMSG with the correct key for each field that requires help. You can specify anything for this key as long as you add the help message to the help database using the same key.
- In Option 4, use **UP SD** (SCRN PARM) to verify that HOLD=Y and HELP=Y. This must be consistent across an application, even though some programs do not use help. HOLD=Y is mandatory to support HELP.

In Option 4, update the SD screen DG parameter as follows for database HLDDBDV1:

Name	Usage	SEGKEY
LTERM	HOLD	IOPCB-LTERM
XFERA	HOLD	
XFERB	HOLD	
XFERC	HOLD	

You do not need to update database HLPDBDV1 since the help program seeking help accesses it.

The total of HOLD segments cannot be less than HOLD AREA + SPA + SCREEN-IMAGE-AREA.

- Add the required transfer work area fields (XFERWKA) as needed.

Setting Up the Help Panels

There are two methods for adding help messages to the HELP database. You can generate and use the following sample Help Maintenance System source modules on pdsqual.SOURCEC or pdsqual.SOURCEP. Or you can use these as a base to create your own unique Help Maintenance application:

COBOL	PL/I	Program Type
HPCTxM	HPPTxM	Menu Screen
HPCTxA	HPPTxA	Add help

COBOL	PL/I	Program Type
HPCTxD	HPPTxD	Display help
HPCTxL	HPPTxL	List help
HPCTxU	HPPTxU	Update help

- HELP key length equals whatever the installation provides (must be same length as HELPMMSG parameters used in the calling program's panel definition)
- One complete help screen of 15 lines goes into one Help database segment by an input SEGLOOP program (see HPCTxA or HPPTxA)
- For screen level help (PF1), the help database key might be hhnnnn (i.e., the calling program's program name), so set up a segment with a key of hhnnnn on the help database

Summary

The following steps summarize how to set up the HELP/HOLD facilities.

1. Put help and hold fields in the transfer work area.
2. Set up the HELP and HOLD databases.
3. Turn on HELP and HOLD in the TDF (use profile defaults or update from the SD screen).
4. Specify the HELPMMSG=**key** of the segment on the Help database (which contains HELP information for that field or screen) in the TDF.
5. Use the help maintenance system add screen (input SEGLOOP processing, 15 lines, 70 bytes each) to add help panels. See pdsqual.SOURCE (HPCTxA) or pdsqual.SOURCE (HPPTxA)
6. Create the help display program, which transfers control from the calling program to display the message (training system uses TRxxVH). This program requires special PF-key processing to return from the help screen.
7. Provide logic for screen level help in PFK1 processing (if implementing), PDSCOPY in member TRPFK1 from pdsqual.SOURCEx.
8. See the examples HPxTxA, etc.

CA Telon-Supplied HELP Program

CA Telon supplies TRCTxH for COBOL applications and TRPTxH for PL/I applications. These help programs:

- Use PF3 to return to the calling program and PF7 and PF8 for paging through selected Help database keys
- Loop back on themselves if you press only Enter, NEXTPGM = **HELP**
- Check by custom code entry point, OINIT2, to ensure a help page exists for the help key

```
If HELP-STATUS-CODE='GE'
  MOVE '** no help available for this field **' TO
  TPO-ERRMSG1
  MOVE SPACES TO IOA-HELP-SEGMENT.
```

- Set the SCRNM PARMS: HELP = **N**, HOLD = **Y**, CAPS = **ON**
- Set the following parameters in data group:
 - All application databases in PCB, REQUEST=@**DUMMY** on all segments
 - HELP database: REQUEST=**OUTREAD**, KEY/WHERE=msg.key
IGNORE=GE
 - HOLD database: REQUEST=**HOLD**, KEY/WHERE=**IOPCB- LTERM**
 - XFERA database: REQUEST=**HOLD** (however many segments of XFERA, B, C are needed)
 - WORKSPA database: as usual

Writing Non-Screen Transactions

The basis of writing non-screen MPPs is to create a standard CA Telon-screen transaction, but suppress the display of the screen by manipulating CONTROL-INDICATOR. Any processing required can be performed in the usual CA Telon manner.

A step-by-step outline follows. It is only an outline because the details differ depending on the processing required.

- Paint a PI. The only required field is ERRMSG1, but you can use the screen area as an intermediate area for automatic mapping.
- Create a PD. This does not actually contain anything other than ERRMSG1, unless the screen area is used for mapping.
- Create an SD. This is set up in the normal way. You must consider the data group and environment screens.

The whole point behind the non-screen transaction is that the normal flow of the CA Telon-generated program is followed. At the relevant point on the output side of the program, PROCESS-INPUT-LIT must be moved to CONTROL-INDICATOR. The relevant point could be OINIT1 if no processing is performed on the output side, OUTTERM if output processing is performed, or any other suitable custom code point.

On the input side of the program, the main consideration is terminating the transaction. You can invoke another transaction using a message switch or move spaces to the SPA to end this conversation. For further information, see "Exiting IMS Application Programs" earlier in this appendix.)

Depending on the processing required, you must insert code at the relevant points.

IMS/DC Report Processing

Often, an online system must transmit report information to a hard-copy terminal. You can define such a report with CA Telon similarly as you define a display screen. Define the report format (using only LITERAL and OUTPUT fields) and create a report definition similarly as you create a screen definition. The major differences between the use of screens and reports within CA Telon are as follows:

- A report definition includes a REPORT statement in place of a SCREEN statement.
- OUTPUT fields are the only variable fields that you can define in a report definition (since no input from a hard-copy terminal is allowed).
- The report definition is not restricted to CRT terminal sizes. It can be any size that is valid for the printer terminal.
- A report program runs as a subroutine under a higher-level program (usually a screen program) rather than as a stand-alone transaction-driven program.

The rest of this section describes the report definition and provides information on the report program structure, controlling the destination of a report, and the interface with the calling program.

Report Definition

As discussed above, a report definition is similar to a screen definition.

CA Telon generates a REPORT statement (instead of a SCREEN or DRIVER statement) to indicate that the definition is for a report.

The parameters for the REPORT statement are similar to those for the SCREEN statement. The REPORT statement has one additional parameter, LINKWKA. LINKWKA is used to pass a user work area from the calling program to the print subroutine, allowing the subroutine access to the data used in the report. LINKWKA is discussed further later in this section.

The only valid generation statements in a report definition are IMSPGM and IMSMFS. TSOPGM is not valid since there is no special TSO test version of a report program. In this case, any TP call (inserted to the message queue for the printer terminal) is traced (a trace screen can be requested), but is not actually executed (that is, nothing is written). This allows the I/O area inserted to IMS to be reviewed in testing for accuracy. To get a layout of the report, however, the program must be run under IMS.

In a report definition, the only valid parameters on the IMSPGM statement are TRACE, GENPCBS, LNKCOPY, USGCOPY, PGMNAME, and MFSMOD.

A TPPCB statement must be included in the report definition, unless the XFER-PCB (the modifiable PCB automatically generated for CA Telon IMS programs) is used to define the terminal destination for the report. Report destination considerations are discussed further below under "Controlling the Report Destination".

Paging cannot be requested for a report (that is, you cannot specify PAGE=Y on the SEGLOOP statement). If paging is necessary, the report program must be called once for each page. For further information, see "Calling Program Interface."

The following screens show the report definition for report PRNT. Note that custom code is added (OINIT1=LTERMINT) to set the print terminal destination based on information in the transfer work area. Since the transfer work area contains all data passed to the report, the LINKWKA parameter is not specified on the REPORT statement. A TPPCB named PRINT is included to define the destination of the report (indicated by the PRINT=Y parameter). You can modify the destination since no LTERM is specified.

Report Program Structure

The structure of the report program is very similar to the output side of a screen program. The main difference is that the report program contains no cursor positioning logic. Parameters on the REPORT statement can be used to add custom code: OINIT1 and OINIT2 to add custom code to the A-100 sections; OCUST1, OCUST2, and OCUST3 to add loop-processing logic to the B-100 section; and PGMCUST to add further custom code to any section.

```
REPORT PRNT,HEADER=TR,XFERWKA=TRXFERWK,OINIT=LTERMINT
TPPCB PRINT,EXPRESS=Y,PRINT=Y
*****
* EMPLOYEE DATA BASE *
*****
DATABAS DBDNAME=TRGDBD01,KEYLEN=16,PCBNAME=EMPLOYEE
SEGMENT BROWSE,DBSEG=TRGEMPL,IMSKEY=TRGEMKEY,KEYLEN=06,
PARENT=0,SEGKEY=EMPL-KEY
*****
* LITERALS *
*****
FIELD LITERAL,POS=(01,15),TEXT='T E L O N T R A I N I
N G S Y S T E M'
FIELD LITERAL,POS=(02,28),TEXT='EMPLOYEE LIST'
FIELD LITERAL,POS=(03,02),TEXT='PAGE'
FIELD LITERAL,POS=(06,02),TEXT='SEQ ID'
FIELD LITERAL,POS=(06,15),TEXT='NAME'
FIELD LITERAL,POS=(06,42),TEXT='SEQ ID'
FIELD LITERAL,POS=(06,55),TEXT='NAME'
FIELD LITERAL,POS=(06,82),TEXT='SEQ ID'
FIELD LITERAL,POS=(06,95),TEXT='NAME'
*****
* VARIABLES *
*****

DATE2 FIELD OUTPUT,POS=(01,73),LTH=008,FMTCTL=MFS
TIME FIELD OUTPUT,POS=(02,73),LTH=008,FMTCTL=MFS
PAGENO FIELD OUTPUT,POS=(03,07),LTH=002,FLDTYPE=NONE
MORE FIELD OUTPUT,POS=(04,02),LTH=007,FLDTYPE=NONE
SEGLLOOP OUTPUT,INCR=(2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2),
LINECNT=SEQ,PAGE=N,CINCR=(40,40)
SEQ FIELD OUTPUT,POS=(09,02),LTH=002,FLDTYPE=NONE,PIC='Z9'
ID FIELD OUTPUT,POS=(09,07),LTH=006,DBNAME=EMPL-KEY
NAME FIELD OUTPUT,POS=(09,15),LTH=020,DBDNAME=EMPL-NAME
CITY FIELD OUTPUT,POS=(10,02),LTH=024,DBDNAME=EMPL-CITY
STATE FIELD OUTPUT,POS=(10,30),LTH=002,DBDNAME=EMPL-STATE
SEGEN
IMSPGM PGMNAME=TRPRINT,TRACE=N
IMSMFS
END
```


In a report program, only three sections are executed from the main section:

- A-100, to perform output initialization including reading data base segments and executing OINIT1 and OINIT2 code
- B-100, to perform output editing, including SEGLOOP processing
- C-200, to write a message to the terminal

Controlling the Report Destination

The destination (i.e., the hard-copy LTERM name) of a report can be defined either as a constant or as a variable.

Note: Specify the destination of a report by indicating the teleprocessing PCB to be used. The report is routed to the hard-copy terminal whose logical terminal name (LTERM) is associated with that PCB. The characteristics of a PCB (including the LTERM associated with the PCB) are determined by the PSB used by the program. In report processing, no PSB is generated for the program; instead, the report uses the PSB of the calling program. Therefore, the LTERM parameter on the TPPCB statement is not valid for a report definition.

To define the report destination as a constant, a TP PCB must be defined and passed to the print subroutine (see "Calling Program Interface" below). This must be a non-modifiable TP PCB whose destination was defined upon generation of the PSB for the calling program. To indicate to the report which TP PCB is used to define the print destination (other TP PCBs can be defined for other functions, such as handling catastrophic error messages), specify PRINT=Y on the correct TPPCB statement.

To define the report destination as a variable, the report program must use a modifiable TP PCB. This program can be either the XFER PCB, which is passed as a standard in most CA Telon IMS programs, or any modifiable TP PCB defined via a TPPCB statement. If a TPPCB statement with PRINT=Y is included, then that PCB is used as the print destination. Otherwise, the XFER PCB is used.

In either case, you determine the print destination by adding procedural code using the OINIT1 or OINIT2 exits to move the destination LTERM to the CA Telon field PRINT-LTERM-NAME. If this 8-byte field is equal to either spaces or the destination already set in the PCB, no CHNG call is executed. Otherwise, a CHNG call to the value in PRINT-LTERM-NAME is executed. In either case, the message is then inserted to the correct PCB. If you want the message to end in a PURG call, then include code within the OINIT1 and OINIT2 code mentioned above to move a "Y" to the CA Telon variable PRINT-PURGE-INDICATOR.

Calling Program Interface

The report program is executed as a subroutine, usually from a CA Telon screen program. Since the code is generated, it has specific interface requirements which dictate the parameters passed. The contents of the parameter list must be as described below.

Note: In PL/I programs, the print subroutine must be defined in the program (using the WKAREA parameter on the SCREEN statement).

Example:

DCL PRNTSUB EXTERNAL ENTRY;

1. LINK-WORK-AREA – The area defined using the LINKWKA parameter. If LINKWKA is not coded, a DUMMY area must be passed. For COBOL, when a dummy area must be passed, you can call it PRNTSUB-AREA. For PL/I, this parameter must always be coded:

ADDR(PRNTSUB_AREA).

2. SPA-XFER-WORK-AREA – The SPA work area.

For COBOL, this must be coded:

SPA-XFER-WORK-AREA.

For PL/I, this must be coded:

ADDR(SPA_XFER_WORK_AREA).

Note: SPA-AREA cannot be passed due to the variance in the length of the SPA-HEADERS.

3. IO-PCB – The IO PCB for the report definition.
4. XFER-PCB – The PCB for the transfer work area.
5. TPPCBs – The teleprocessing PCBs in the report definition, if any.
6. DBPCBs – The database PCBs in the report definition, if any.

Note: The order of the TPPCB and DATABAS statements in the report definition determines the order of the TPPCBs and the DBPCBs.

A sample call when no work area is defined using the LINKWKA parameter and a single TP PCB used for the report destination is shown below:

- For COBOL:

```
CALL 'TRIMPRNT' using PRINTSUB-AREA  
SPA-XFER-WORK-AREA  
IO-PCB  
XFER-PCB  
PRINT-PCB  
EMPLOYEE-PCB.
```

- For PL/I:

```
CALL TRIMPRNT (ADDR(PRINTSUB_AREA) .  
ADDR(SPA_XFER_WORK_AREA) .  
IO_PCB  
XFER-PCB  
PRINT-PCB  
EMPLOYEE-PCB.
```

CA Telon Driver Applications

To use multiple copy members that contain the PCB mask definition and the PCB list for applications using multiple DRIVERS, you could:

- Set up different compile JCL for each DRIVER so these copy members are picked up from different libraries
- Use the APPLID parameter available at the SCREEN or DRIVER level to identify these copy members at a sub-application level

Example:

- Use the APPLID to identify a subsystem
- Change the PCB mask and PCB list copy member names in the pdsqual.MACLIB member PGMNAMES to include APPLID
- Create multiple copy members that satisfy the sub-application's requirement

CA Telon Program Transfers

This subject presents some program transfer scenarios and summarizes the CAs/disCAs of these different types of program transfers.

Message - Switch Transfer

You enter the terminal Tran-Code or /Format. IMS schedules the transaction and loads the program and PSB. Program one executes output processing, sends an image (message), and requests more work (reads the message queue).

You enter data or use a function key. IMS schedules the transaction and loads the program and PSB. Program one executes input processing and message switches to the next program. IMS schedules the transaction and loads the program and PSB. Program two (a new program) executes for output processing, sends an image (message), and requests more work (reads message queue).

You enter data or use a function key.

CAs

- One transaction ==> one PSB==> one CA Telon program
- Fairly small load modules
- Facilitates transfers to non-CA Telon programs

DisCAs

- Large number of programs and PSBs defined to IMS
- Processing overhead due to extra transaction schedule per terminal I/O

CA Telon Dynamic-Link Transfer

You enter a terminal Tran-Code or /Format. IMS schedules the transaction and loads the program and PSB. Program one executes output processing, sends an image (message), and requests more work (reads message queue).

You enter data or use a function key. IMS schedules the transaction and loads the program and PSB. Program one executes input processing and calls the next program. Program two executes for output processing, sends an image, and returns to program one.

Note: No PSB load occurs; program two uses one's PSB.

You enter data or use a function key.

CAs

- One transaction ==> one PSB ==> CA Telon program
- Fairly small load modules
- One IMS transaction schedule per terminal I/O

DisCAs

- Requires compatible PSB structure between input processing of Program A with the PSB structure of Program B, if Program A can transfer to Program B
- Two OS loads per terminal I/O

CA Telon Static-Link Transfer

You enter terminal Tran-Code or /Format. IMS schedules the transaction and loads the program and PSB. The driver program initializes and calls program one. Program one executes for output processing, sends an image (message), and returns control to the driver. The driver requests more work (reads the message queue).

You enter data or use a function key. IMS schedules the transaction and loads the program and PSB. The driver program initializes and calls program one. Program one executes for input processing, indicates program two is the next program, and returns control to the driver. The driver calls program two. Program two executes for output processing, sends an image (message), and returns control to the Driver. The Driver requests more work (reads the message queue).

You enter data or use a function key.

CAs

- One transaction ==> one PSB ==> all CA Telon programs for an application
- Best performer if the program is preloaded
- Minimizes memory requirements when preloaded
- Can use customized drivers to improve the interface of CA Telon and non-CA Telon systems

DisCAs

- Very large load modules
- Long load time if not preloaded
- No individual IMS statistics per screen, as all screens are part of the one transaction
- Changes to any screen programs require a relink of all programs under the driver
- One PSB for the entire application, total compatibility required

CA Telon Static/Dynamic-Link Transfer

You enter terminal Tran-Code or /Format. IMS schedules the transaction and loads the program and PSB. The driver program initializes and calls program one. Program one executes for output processing, sends an image (message), and returns control to the driver. The driver requests more work (read the message queue).

You enter data or use a function key. IMS schedules the transaction and loads the program and PSB. The driver program initializes and calls program one. Program one executes for input processing, indicates program two is the next program, and returns control to the driver. The driver calls program two. Program two executes for output processing, sends an image (message), and returns control to the driver. The driver requests more work (reads the message queue).

You enter data or use a function key. IMS schedules the transaction and loads the program and PSB. The driver program initializes and calls program two. Program two executes for input processing, indicates program three is the next program, and returns control to the driver. The driver calls program three. Program three executes for output (no PSB load), sends an image (message), and returns control to the driver.

CAs

- Good performer if preloaded
- All commonly used programs in one module
- All lesser used programs outside the module, reducing load module size
- Increased flexibility
- Facilitates transfer to non-CA Telon programs

DisCAs

- No individual IMS statistics from screens within the static load module
- Changing any screens within the static load module requires complete relink
- Compatible PSB structure required between static load module and dynamically linked programs

Program Switching to Non-CA Telon Programs Using MFS

This method switches from a CA Telon program to a non-CA Telon program that uses MFS for program switching. This method is designed for CA Telon static programs running under a driver without data being passed. In the IMS static program (not the CA Telon driver):

PL/I

```
IF PFKEY_INDICATOR = 5 (see note 1)
  THEN DO;
    CONTROL_INDICATOR = TRANSACTION_COMPLETE_LIT;
    LINKAGE_OUTPUT_MODNAME = 'modname'; (see note 2)
    LINKAGE_WRITE_XFER_INDIC = 'I';
    TPO_LLZZ_BUFSIZE = 5;
    TP_OUTPUT_BUFFER_FIELDS = SPACES;
  END;
```

COBOL

```
IF PFK5-17 (see note 1)
  MOVE TRANSACTION-COMPLETE-LIT TO CONTROL-INDICATOR
  MOVE 'modname' (see note 2)
  TO LINKAGE-OUTPUT-MODNAME
  MOVE 'I' TO LINKAGE-WRITE-XFER-INDIC
  MOVE 5 TO TPO-LLZZ-BUFSIZE
  MOVE SPACES TO TP-OUTPUT-BUFFER-FIELDS.
```

Note:

- Using a PF key is only one method of determining when to trigger the switch from one program to another
- modname should be the name of the MOD you want to insert
LINKAGE-WRITE-XFER-INDIC of **I** was arbitrary. You can use anything except the **X**, **M**, or **W** characters used by CA Telon.

In the IMS CA Telon DRIVER (custom code point TERM):

PL/I

```
IF LINKAGE_WRITE_XFER_INDIC = 'I'
  THEN DO;
    CALL C_970_PUT_MESSAGE;
    LINKAGE_WRITE_XFER_INDIC = ' ';
  END;
```

COBOL

```
IF LINKAGE-WRITE-XFER-INDIC = 'I'  
  PERFORM C-970-PUT-MESSAGE  
  MOVE SPACES TO LINKAGE-WRITE-XFER-INDIC.
```

Using Multilingual MFS Screens

You must address the following areas when implementing a system using multilingual screens:

- Help messages
- Screen literals
- Error messages
- hhWKAREA (standardized messages)

Each user signs on with a transaction identification code directly associated with a two-character language code in a table. This language code is then moved to a PIC XX field in the transfer work area.

HELP Messages

In the PD editor for the screen, put a **U** against each field you want to invoke the HELP facility, or select the HELPMMSG option from the top of the PD screen. On the update screen for each of these fields, put in the HELPMMSG field key (key to the HELP database). This key is always the same for a particular field, no matter what the language.

When the normal **?** method passes the key to the HELP program, the help program picks up the two-byte prefix from the transfer work area and concatenates it with the help key. This creates a language specific access to the appropriate error message.

Screen Literals

Have your CA Telon coordinator open up a custom code entry point in the IMS program at C-200T using PGMCUST. This is achieved by updating the TELONIIS member on pdsqual.MACLIB and setting PGMCUST = **C200T**. It can be performed on an application by application basis.

In the program itself, have a standard piece of custom code at C-200T to read the prefix from the transfer work area and change the MODname (write various MODs in the different languages) before sending the message to the terminal.

Error Messages

Put all error messages into a table, one table per language for each message. The search argument for this table is the field name of the field in error prefixed by the two-character, language-specific code.

Write a XFEDIT with SRC code to access the table and move the appropriate error message to TPO-ERRMSG1.

Set up a work area that looks like hhWKAREA for each language and define them on the SD.

In custom code entry point OINIT1 code, enter:

```
MOVE prefix-workarea TO hhWKAREA.
```


Appendix B: Using the CICS Online Environment

This appendix discusses only topics unique to the CICS online transaction processing environment.

This section contains the following topics:

[Handling the Clear Key](#) (see page 595)

[Invoking a Nonterminal Program](#) (see page 596)

[Using Temporary Storage Instead of DFHCOMMAREA](#) (see page 596)

[CA Telon Program to/from Non-CA Telon Program](#) (see page 597)

[Accessing a Dynamically Loaded Table](#) (see page 599)

[CICS Driver Program](#) (see page 599)

Handling the Clear Key

CICS application programs often require that the application program resend the output screen when you press the Clear key.

Because CA Telon optimizes the output data stream, just detecting Clear and setting CONTROL-INDICATOR to DO-WRITE-LIT is not enough. The following code ensures that all the output data stream is resent to the terminal.

```
IF CLEAR
  MOVE DO-WRITE-LIT TO CONTROL-INDICATOR
  MOVE SCREEN-IMAGE TO TP-OUTPUT-BUFFER-FIELDS
  MOVE LOW-VALUES TO SCREEN-IMAGE-AREA
  GO TO P-100-PFKEYS-RETURN
```

After testing the above code, you might want to incorporate it automatically at the start of P-100 in all CICS programs by using the PGMSTD parameter in the TLNIIS macro.

Note: CA Telon already generates code to return to CICS with no transaction ID if you press the Clear key. You must include the above code, therefore, at the start of P-100.

Invoking a Nonterminal Program

The nonterminal program can be tested by transferring control to it through an 'XCTL'. This is done by the CA Telon default 'DO-TRANSFER (value of NEXT-PROGRAM-NAME-ID) at the end of input processing. The report is sent to the screen.

After testing the transfer to the nonterminal program, you may test in native CICS, with an actual EXEC CICS statement to start the nonterminal transaction. The report will be sent to the CICS printer defined.

Example:

```
EXEC CICS START
  TRANSID (WS-NONTERM-TRAN-ID)
  TERMID (WS-NONTERM-TERM-ID)
END-EXEC.
```

Using Temporary Storage Instead of DFHCOMMAREA

Benefits from using CICS temporary storage versus the CA Telon DFHCOMMAREA implementation are two-fold:

- First, and most importantly, CICS installations, using the Multi-Region Option (MRO) facility in CICS, generally define one Terminal Owning Region (TOR). The TOR communicates with one or more Application Owning CICS Regions (AOR).
- In MRO configurations as described above, the CICS communication area (COMMAREA) is assigned to terminal storage and resides in the region connected to the terminal. This results in all COMMAREAs residing in the TOR, rather than in the AOR where they are used. This creates unnecessary overhead in the TOR.
- Secondly, for those installations running OS/390 or z/OS CICS 1.6 (or above), temporary storage can be located above the 16M line, thus freeing up storage below the line for execution of application programs. With PL/I 1.5.1 and COBOL II, the COMMAREA can also be located above the line; in this case, the benefits derived from using temporary storage are nullified.

The implementation of the temporary storage feature from an external point of view was a modification to the TLNIIIS/SETENV parameter SPASTG. This implementation uses Temporary Storage Scratch Pad Areas (SPAs) on an application level but not on the program level (which is not practical and is not supported).

Specification of the use of temporary storage for transfer information is done by setting SPASTG =**AUTO,TSM**AIN.

Note: This is only an example, you can also use STATIC SPAs with temporary storage. Code the temporary storage specification as either TSMAIN for main temporary storage or TSAUX for auxiliary temporary storage. Please refer to the CA Telon MACRO member TLNIIS in the CA Telon pdsqual.MACLIB data set for further information.

CA Telon Program to/from Non-CA Telon Program

Transfers COMMAREA compatibility is the only area of concern when you transfer from a non-CA Telon CICS online program to a CA Telon CICS online program by an XCTL command. The COMMAREA defined to the CA Telon program must be large enough to hold the COMMAREA as defined in the non-CA Telon program within the transfer work area.

The layout of the CA Telon COMMAREA is:

- Sixteen-byte header with COMMAREA length, next program name, and last transaction code to execute
- User transfer work area
- Screen image area

Thus, the size of the CA Telon program's COMMAREA is at least 16 + size of non-CA Telon program's COMMAREA + size of the largest screen image area for the CA Telon application (all programs with the same header).

The first processing that must occur when entering the CA Telon program from a non-CA Telon program with a COMMAREA is the reformatting of the non-CA Telon COMMAREA. To do this:

- Create storage to hold the non-CA Telon COMMAREA data during the reformatting process. This can be static program storage or CICS Temporary Storage and is referred to as OLD-COMMAREA-DATA. You should use temporary storage only when the size of the old COMMAREA would result in an overly large program if a copy of the program were placed in static storage (when additional custom code is required to access temporary storage).
- Define PGMCUST = MAINI,CAFORMAT
- Create CAFORMAT custom code as follows:

```
IF EIBCALEN = "length of non-CA Telon COMMAREA"
MOVE SPA-AREA TO OLD-COMMAREA-DATA
MOVE "length of CA Telon COMMAREA" TO SPA-LENGTH
MOVE CURRENT-PROGRAM NAME TO NEXT-PROGRAM-NAME
MOVE 'xxxx' TO SPA-TRANSACTION-CODE
MOVE OLD-COMMAREA-DATA TO SPA-XFER-WORK-AREA
MOVE DO-TRANSFER-LIT TO CONTROL-INDICATOR
PERFORM MAIN-PROCESS.
```

When returning to non-CA Telon programs:

- Define PGMCUST = **C300I,NONEXCTL**
- Create NONTXCTL custom code as:

```
IF NEXT-PROGRAM-NAME = "non-CA Telon program"
EXEC CICS XCTL PROGRAM(NEXT-PROGRAM-NAME)
COMMAREA(SPA-XFER-WORK-AREA)
LENGTH("LENGTH OF non-CA Telon COMMAREA")
END-EXEC.
```

If the non-CA Telon program transferring into a CA Telon program does not pass a COMMAREA, there is no special code required in the CA Telon program.

However, when a CA Telon program must XCTL to a non-CA Telon program without a COMMAREA, custom code to perform the XCTL is required, as follows:

- Define PGMCUST = **C300I,NOCAXCTL**
- Create NOCAXCTL custom code member as:

```
IF NEXT-PROGRAM-NAME = "non-CA Telon program"
EXEC CICS XCTL PROGRAM(NEXT-PROGRAM-NAME)
END EXEC.
```

Accessing a Dynamically Loaded Table

To access a dynamically loaded table called TABLE01, modify the custom code as follows:

```

EDIT —— TRCC2L.SD BLLPTRS MEMBER 001 OF 003 SIZE 000018 COL 07
COMMAND ==> SCROLL ==> CSR
=MBR=> TRCC2L.SD(BLLPTRS) 000007
000001 *****
000002 ***** BLL POINTERS TO USER SPECIFIED AREA
000003 *****
000004 .....
000005 .....
000006 05 BLL-TABLE01-POINTER1 PIC S9(8) COMP.
000007 * (ONE BLL POINTER FOR EACH 4096 BYTES)
=MBR=> TRCC2L.SD(INITPROC) 000004
000008 .....
000009 EXEC CICS LOAD PROGRAM(TABLE01)
000010 SET (BLL-TABLE01-POINTER)
000011 END-EXEC.
=MBR=> TRCC2L.SD(USRAREA) 000007
000012 *****
000013 ***** USER SPECIFIED AREAS FOR SPECIFIC CICS PROCESSING
000014 *****
000015 01 TABLE-01-LAYOUT.
000016 :
000017 : (LAYOUT OF DYNAMICALLY ACCESSED TABLE
000018 :
***** ***** BOTTOM OF DATA *****

```

CICS Driver Program

The driver concept as applied to CA Telon-generated CICS programs uses one transaction code for an entire application. This means only one transaction code is added to the PCT table, minimizing table maintenance. In addition, this saves processing time by minimizing table searches.

The logic for the driver is simple and consists of a check of the EIBCALEN to decide which program to transfer control. If the EIBCALEN is zero, the driver executes a GETMAIN and then transfers to the first program in the application. If the EIBCALEN is not zero, the driver transfers to the SPA-NEXT-PROGRAM-NAME in the COBOL version and COMMAREA-NEXT-PROGRAM-NAME in the PL/I version. The adjustments to the generated programs are:

- Include custom code in the C-300I Custom code point for PL/I or COBOL:

```
SPA-TRANSACTION-CODE = '****'; (for PL/I)
- or -
MOVE '****' TO SPA-TRANSACTION-CODE. (for COBOL)
```

The asterisks in SPA-TRANSACTION-CODE cause output processing in the program being transferred to because SPA-TRANSACTION-CODE does not equal PROGRAM-TRANSACTION-CODE. This is checked in the MAINLINE portion of the CA Telon-generated programs.

- Use the TRANCDE parameter on the Update CICS Screen Environment screen to set PROGRAM-transaction-code in each of the CA Telon-generated programs to the driver's transaction code.

The PROGRAM-TRANSACTION-CODE set to the driver's transaction code always returns control to the driver after output processing. The driver then transfers to the input side of the CA Telon-generated program, which performs input processing, and transfers control to the output side of SPA-NEXT-PROGRAM-NAME.

Appendix C: Using The Batch Environment

This appendix addresses the development of application programs destined for the batch target environment and discusses processing passed parameters and batch design considerations.

Processing Passed Parameters

Parameters can pass to CA Telon batch programs either from other application programs (online or batch) or directly from a JCL EXEC card.

Sample Code

- PL/I:
`CALL 'batchpgm' (WS_PARM1, WS_PARM2, WS_PARM3);`
- COBOL
`CALL 'batchpgm' USING WS-PARM1 WS-PARM2 WS-PARM3.`
- JCL
`//STEP1 EXEC PGM=batchpgm,PARM=(parm1,parm2,parm3)`

In batch programs, you can use the fields at the bottom of the TDF BD screen to address incoming parameters.

Example:

If your program is passed three parameters with lengths respectively six bytes, eight bytes, and four bytes, you simply type on the bottom line:

LINKAGE: PARM5_6_8_4_____

You can then reference these fields in your program as:

- **PRM-FIELD-1 (PRM_FIELD_1)**
- **PRM-FIELD-2 (PRM_FIELD_2)**
- **PRM-FIELD-3 (PRM_FIELD_3)**

The CA Telon generator automatically generates the correct code to place the passed parameters into these fields. They are defined (declared) as PIC X (CHAR) fields.

Example:

If PRM-FIELD- is really a PIC 9(7) COMP-3 (FIXED DEC(7,0)) field, then you code the following in the program's WKAREA custom code entry point:

In PL/I programs

```
DCL WS_DECIMAL_PARAMETER  
  BASED(ADDR(PRM_FIELD_3)) FIXED DEC(7,0);
```

In COBOL programs

```
01 WS-DECIMAL-PARAMETER PIC 9(7) COMP-3.  
01 WS-CHARACTER-PARAMETER REDEFINES  
  WS-DECIMAL-PARAMETER PIC X(4).
```

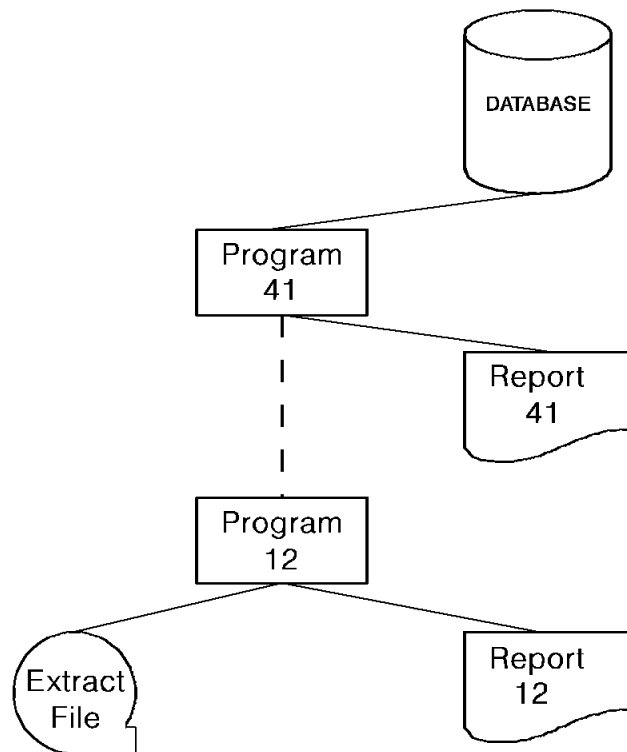
COBOL programmers must then code the following statements in the ININIT1 custom code entry point (or ININIT for pre-release 2.1 sites):

```
MOVE PRM-FIELD-3 TO WS-CHARACTER-PARAMETER.
```

Batch Design Considerations for Packaging Reports

- Identify controlling module
- Have controlling module call other module(s) as subroutine(s). for example:
 - Report 41 does a sequential read of the database
 - Report 41 calls Report 12 each time it reads a root segment
 - Modify mainline section of Report 12

Example Batch Program



VSAM Example

In program TRxx41, custom code entry point GETTRAN:

```

IF TRGEMPLB-EOF
  MOVE END-TRAN-LIT TO CONTROL-INDICATOR
  CALL 'TRBPxx12' USING
    IOA-TRGEMPLB-SEGMENT
    BWA-TRANSACTION-COUNT
    CONTROL-INDICATOR.
  
```

In custom code entry point PRCTRAN:

```

IF RECORD-TYPE = '1'
  CALL 'TRBPxx12' USING
    IOA-TRGEMPLB-SEGMENT
    BWA-TRANSACTION-COUNT
    CONTROL-INDICATOR.
  
```

IMS Example

In program TRxx41, custom code entry point GETTRAN:

```
IF EMPLOYEE-STATUS-CODE = 'GB'  
  MOVE END-TRAN-LIT TO CONTROL-INDICATOR  
  CALL 'TRBPxx12' USING  
    IOA-TRGEMPL-SEGMENT  
    BWA-TRANSACTION-COUNT,  
    CONTROL-INDICATOR.
```

In custom code entry point PRCTRAN:

```
IF EMPLOYEE-SEGNAME-FB = 'TRGEMPL'  
  MOVE IOA-GENERIC TO IOA-TRGEMPL-SEGMENT  
  CALL 'TRBPxx12' USING  
    IOA-TRGEMPL-SEGMENT  
    BWA-TRANSACTION-COUNT,  
    CONTROL-INDICATOR.
```

Program TRxx12 becomes a subroutine of program TRxx41. To achieve this:

- Change the mainline so it is no longer a loop
- Take out the database call (done in TRxx41)
- Take out C-1000
- Execute B-2000 and T-1000 only if the control flag indicates end of input transactions

What needs to be done in TDF:

- Update BD (TDF option 5):
PGMCUST = (MAINLINE,MAINLINE)
- Update ENvironment:
LNKCOPY = LNKCODE
USGCOPY = USGCOPY
- Enter the following custom code for LNKCODE:
01 IOA-TRGEMPL-SEGMENT.
COPY TRGEMPL.
01 WS-BWA-TRANSACTION-COUNT PIC 9(7) COMP-3.
01 WS-CONTROL-INDICATOR PIC X(12).

Note: You cannot call the second and third parameters.

BWA-TRANSACTION-COUNT and CONTROL-INDICATOR, because these names are automatically generated in TRxx12 Working Storage.

IOA-TRGEMPL-SEGMENT, however, is not generated automatically in Working Storage because TRxx12 contains no automatic database I/O.

Enter the following custom code for USGCODE:

```
IOA-TRGEMPL-SEGMENT,  
WS-BWA-TRANSACTION-COUNT,  
WS-CONTROL-INDICATOR.
```

Enter the following custom code for MAINLINE:

```
MOVE WS-CONTROL-INDICATOR TO CONTROL-INDICATOR.  
MOVE WS-BWA-TRANSACTION-COUNT TO  
  BWA-TRANSACTION-COUNT.  
IF BWA-TRANSACTION-COUNT = 1  
  PERFORM Q-1000-PROGRAM-INIT. (open report 12 file)  
  IF CONTROL-INDICATOR = 'TRANSACTEND'  
    (transaction complete)  
    PERFORM B-2000-END-REPORT  
    PERFORM T-1000-PROGRAM-TERM  
  ELSE  
    PERFORM A-1000-PROCESS-TRAN  
    PERFORM B-1000-PROCESS-DETAIL.  
GOBACK.
```

Note: Once you set up and test this mechanism, you can use the same custom code (**LNKCODE,USGCODE,MAINLINE**) for most subroutines.

Appendix D: Using The DB2 Environment

This appendix discusses the IBM relational database DB2 environment and includes the following:

- Catalog access interface
- Using SEGEDITs against DB2 tables
- Linking considerations
- BROWSE request
- Plan name qualification
- Executing CA Telon developed applications
- DB2 Get Diagnostics support
- Fetch Details—Using an alternate cursor
- Fetch Details—FETCH for NN ROWS
- User data types
- Declaring global temporary tables

For further information on using the DB2 options, refer to the *Data Administration* manual.

This section contains the following topics:

[Catalog Access Interface](#) (see page 608)

[Using SEGEDITs Against SQL Tables](#) (see page 612)

[Linking Considerations](#) (see page 613)

[BROWSE Request](#) (see page 617)

[Plan Name Qualification](#) (see page 617)

[Executing CA Telon-Developed Applications](#) (see page 624)

[FETCH Details—Using an Alternate Cursor](#) (see page 630)

[FETCH Details—FETCH for NN Rows](#) (see page 633)

[User DATATYPES](#) (see page 634)

[Declaring Global Temporary Tables](#) (see page 634)

Catalog Access Interface

The CA Telon Design Facility provides dynamic access to the DB2 catalog through the DB2 Catalog/Import screen and the DB2 interface module TNMCDB2I. The CA Telon Design Facility makes a service request to DB2I, which then performs the necessary access against the catalog. To understand what requests are made and how DB2I must respond, first examine what information CA Telon can use from the DB2 catalog and how the catalog is structured.

Note: For the PWS, a batch DB2 catalog extract system is provided to allow the extraction of DB2 table information in the form of a transport file. This system is provided in both COBOL and PL/I.

Information Used by CA Telon

CA Telon stores information about DB2 tables and views. This information is accessed using the actual DB2 table or view name and by a CA Telon short name. The DB2 name is a two-part name, composed of the DB2 creator and table/view name. The CA Telon short name is an eight-character name unique among all DB2 table/view definitions defined to CA Telon. Other information stored in the TDF relating to a DB2 table or view is the:

- Name of the DCLGEN member for this table/view
- Names of all columns defined in this table/view along with their types, their lengths, and any PL/I or COBOL I/O area variable name associated with the column
- Access indicator for each column variable
- Key or index indicator to identify columns used to construct indices for a table/view and the relative position of the column in the index
- *NOT NULL* indicator for each column
- Copy member information used to define alternative I/O areas

You can enter this information directly into the TDF by requesting a **CR TB** from the TDF Option 2 menu. Alternatively, you can capture most of this information from the DB2 catalog by using the DB2 Catalog/Import screen. To access the DB2 Catalog/Import screen, enter a **CA D2** from the TDF Option 2 menu. The DB2 Catalog/Import screen then presents a list of the DB2 tables and views defined in the DB2 catalog, allowing the user to request an **IMPORT** of items from the list.

DB2 Catalog Structure

The DB2 catalog is a group of DB2 tables. The following tables contain information the TDF can use:

- SYSIBM.SYSTABLES
- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSKEYS

SYSIBM.SYSTABLES

SYSIBM.SYSTABLES contains one row for each DB2 table and view. The TDF can use the following information stored in SYSTABLES:

- Table/view creator
- Table/view name
- Type to indicate if this item is a table or view
- Name of the database containing this table/view
- Name of the tablespace containing this table/view
- Number of columns defined in this table/view
- Remarks associated with this table/view when created

The DB2 Catalog/Import screen presents the list of DB2 tables and views to the TDF user by calling DB2I, once for each line of the list, during its OUTPUT processing. DB2I sequentially accesses the SYSTABLES rows, returning one row's data each time it is called with this request.

Note: The DB2 Catalog/Import screen also accesses the TDF data set TNTDX to indicate which DB2 catalog entries already have associated information stored within the TDF.

Once the DB2 Catalog/Import screen presents this list, you can request that DB2 catalog information be brought into the TDF using one of three line commands:

- **I** imports all DB2 information for the table on that line. This function issues an error if the entry requested already has information within the TDF.
- **O** imports all DB2 information for the entry regardless of the existence of information within the TDF. It results in a complete overlay of any existing information.
- **A** extends existing TDF information relating to a DB2 table or view. The entry imported by this command is not created in the TDF. Rather, its information is added to whatever DB2 table or view you specified for that entry in the CA Telon short name.

SYSIBM.SYSCOLUMNS

The DB2 Catalog/Import screen processing is very similar for all types of imports. First, it requests DB2I to verify that the DB2 catalog is accessible. This request causes DB2I to SELECT the SYSTABLES row for the particular DB2 item being processed. Next, the DB2 Catalog/Import screen requests column information for the DB2 item being processed. DB2I accesses the SYSIBM.SYSCOLUMNS table to retrieve information about a column defined in the DB2 item being imported. SYSCOLUMNS contains one row for each column of each DB2 table and view. The TDF uses the following SYSCOLUMNS information:

- Name of the column
- Name of the table/view containing the column
- Name of the creator of the table/view containing this column
- Number designating the relative position of this column
- Type of data contained in this column
- Length of the data stored in this column
- Scale of this column if it contains decimal data
- Null indicator for this column
- Relative numerical position of column in a primary key

DB2I returns one column's information to the DB2 Catalog/Import screen each time the TDF accesses it for SYSCOLUMNS data. The DB2 Catalog/Import screen continues to access DB2I for SYSCOLUMNS information until DB2I indicates that no more columns exist for the item being imported. The DB2 Catalog/Import screen is ready to move on to the next DB2 item for import, overlay, or add-on processing. The above description defines what the *vanilla* version of DB2I does when the TDF calls it for import, overlay, or add-on processing. You can set up DB2I to perform additional processing based on the existence of index information in the SYSIBM.SYSKEYS and SYSIBM.SYSINDEXES DB2 catalog tables. It can also use REMARKS associated with the table or view to propagate the CA Telon description data (see the remarks in DB2I on the current installation tape for details on implementing these special features).

SYSIBM.SYSINDEXES

SYSIBM.SYSINDEXES contains one row for every index defined to DB2. It contains the following information used by DB2I:

- Name of the index
- Name of the creator of the index
- Name of the table on which the index is defined
- Name of the creator of the table on which the index is defined
- Indicator of whether the index is unique or not

- Number of columns making up the key for this index
- Indicator of whether this index is clustered

SYSIBM.SYSKEYS

SYSIBM.SYSKEYS contains one row for each column that is part of a key for an index. It contains the following information used by DB2I:

- Name of the index that uses this column as a key
- Name of the creator of the index that uses this column as a key
- Name of the column that this row represents
- Relative numerical position of this column in the key
- Relative position of this column in its table
- Indicator of the collating sequence for this column in the key (i.e., ascending or descending)

When using index processing, DB2I checks every DB2 table being processed for a SYSINDEXES row. For each index found, the columns in the DB2 table that make up the index key are marked as key columns. A number indicating the column's relative position within the index key marks these columns. Since each index requires its own numerical sequencing, a different TLNROW is created for each index.

Note: The first index processed is always the UNIQUE-CLUSTERED index, if such an index exists.

When you import a DB2 table, you get a TLNROW for each index defined on the catalog, and a *primary* TLNROW. With no indexes defined, you get the primary **TLNROW** with no KEY/ACC flags set. With one or more indexes defined, the following list describes the order of your TLNROWS (each named by the first eight characters of the index name):

1. The clustered index first (if one exists)
2. Any unique non-clustered indexes (alphabetically by index name, creator)
3. Any non-unique, non-clustered indexes next (alphabetically by index name, creator)

An existing clustered index creates two TLNROWS from this index, both having the same key settings in Option 2. The second TLNROW has 'ACCESS EQUAL NO' on the keyed columns for use in an update situation where the key values are not updated. Using this TLNROW dramatically improves performance. Use the first TLNROW for all other accesses using this index.

Using SEGEDITs Against SQL Tables

You can use a SEGMENT EDIT with an SQL table, not just with DL/I databases. Use a SEGEDIT to determine if a key value entered by the user into an online transaction screen has a corresponding row in an SQL table. The following example uses the CA Telon training SQL table TRGEMPL. Assume that only the column EMPL-ID was designated as the key column in Data Administration.

The example shows how to generate a SEGEDIT that tests for the existence of an employee name (EMPL-NAME), even though this column is not designated as a key field. Perform the following steps:

1. In Option 2, Data Administration, create a new CA Telon Row (TLNROW) with just the EMPL-NAME column. This ensures that the rest of the columns in the table's row are not returned to the program (with the inherent DBMS overhead) every time the SEGEDIT is invoked. To do this, Update the SQL table in Option 2, group copy (using the CC line commands), and edit both the TLNROW and the desired column names to create a new TLNROW. Change the TLNROW name as shown below:

```
UPDATE SQL TABLE ***** TELON.TRGEMPL ***** SIZE 000006
COMMAND ==> SCROLL ==> CSR
TLNNAME TRGEMPL_ DESCR TELON TRAINING EMPLOYEE TABLE _____ SYNONYM _ Y/N
DCLCOPY TRGEMPL_ DCLLBL _____ DCLRDEF _ Y/N
COPY _____ COPYLBL _____ COPYLV1 _ Y/N

COLUMN NAME ALIAS KY/AC TYPE LTH/DEC -NU
*****
000001 —TLNROW-- TRGEMPL —
000002 EMPL_ID EMPL-ID 1 Y CHAR 6 Y
000003 EMPL_NAME EMPL-NAME CHAR 25 Y
000004 EMPL_DOB EMPL-DOB DATE Y
.....
.....
000007 —TLNROW-- EMPLNAME —
000008 EMPL_NAME EMPL-NAME Y CHAR 25 Y
***** ***** BOTTOM OF DATA *****
```

2. In the data group of the screen or batch definition, DGADD TRGEMPL.TAB to pull in the above table definition containing the two TLNROWS.

```
UPDATE DATA GROUP — TRCC2A.SD SIZE 000017 COL 01
COMMAND ==> SCROLL ==> CSR
LABEL REQUEST KEY/WHERE IGNORE
=====
TAB==> TRGEMPL TELON-TRGEMPL
ROW=> TRGEMPL @DUMMY @EMPL-ID
ROW=> EMPLNAME @DUMMY @EMPL-NAME TRGEMPL
***** ***** BOTTOM OF DATA *****
```

3. To generate the correct WHERE clause in our SEGEDIT to use the EMPL-NAME column, fill in the SEGEDIT panel as follows:

```
TRCC2A.PD UPDATE SEGEDIT *****
COMMAND ==> _____
EDIT NAME NAMECHK
COPY EDIT BASE: _____
SEGMENT NAME: EMPLNAME PCBNAME: _____
KEY: TPI-NAME _____
WHEN: _____
_____
_____
_____
ERROR CONDITION: NOTFOUND
HIGHLIGHT FIELDS: NAME_____
ERRCHAR FIELDS: _____
CURSOR FIELD: _____
ERROR MESSAGE: EMPLOYEE NAME NOT FOUND ON DATA BASE _____
_____
CALL FUNC: _____ OPCODE: _____

DLI QUALITY: _ CMDCODE: ____ I/O AREA: _____
SSALIST: _____
_____
VSAM SEGMENT LTH: _____
GEN KEY LTH: _____
```

Note: SEGMENT NAME specifies the newly created TLNROW whose key is EMPL-NAME. KEY specifies the TPI field that contains the value used to access the database.

Linking Considerations

The DB2 pre-compiler replaces SQL statements with a call to DSNHLI using variables from the SQL-PLIST. DSNHLI is the interface or connection between the environment the program is running in (that is, TSO, IMS, and so on) and DB2.

DSNHLI has some aliases. Among these are DSNELI and DFSLI000.

Batch programs

At link-edit time in a batch program, you must make sure that DSNELI is linked. DSNELI is the BATCH to DB2 interface module and is an alias of DSNHLI.

MPP and BMP programs

For MPP and BMP programs, make sure that DFSLI000 is linked. DFSLI000 is the IMS-to-DB2 interface module and is another alias of DSNHLI. To further complicate things, CBLTDLI is also an alias of DFSLI000. By coding entry CBLTDLI, you automatically link DFSLI000 into your program.

MPPs present no problem because their SYSLIB concatenation resolves DSNHLI to DFSLI000 automatically.

Batch versus BMP programs

Most installations use the same procedures to compile and link both Batch and BMPs. SYSLIB, in the link-edit step, has a concatenation that resolves DSNHLI as either DSNELI or DFSLI000. This sometimes causes a DB2 error: -922 Connection Authorization Failure. The problem usually occurs when running a BMP; the library concatenation normally has DSNELI first.

To resolve this problem when linking a batch program, use an INCLUDE SYSLIB (DSNELI) in the link-edit step of your JCL. For a BMP program, use INCLUDE SYSLIB (DFSLI000).

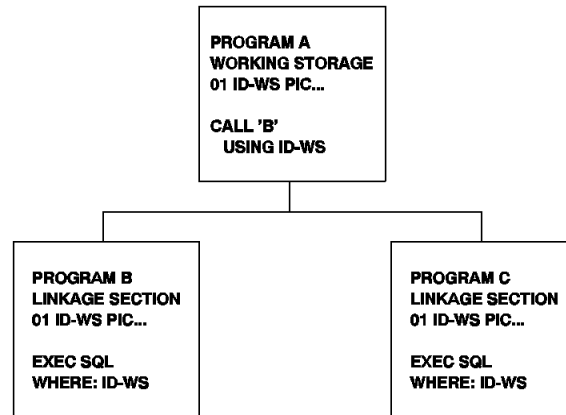
DSNHADDR (COBOL) and SQL-PLIST

CA Telon generates programs with DB2 SQL statements to access DB2 tables. The DB2 pre-compiler interprets these statements and creates an SQL-PLIST for each SQL call. (The SQL-PLIST contains addresses, lengths, and so on, and is used by DSNHADDR.) It also adds source code to the start of COBOL application programs. This source code includes a call to DSNHADDR, which establishes addressability on the host variables used in SQL calls.

DSNHADDR is called only once (controlled by a flag -SQL-INIT-FLAG) when the first SQL call is invoked.

Note: Because addressability is established only once, host-variables used in a WHERE clause should *not* be defined in the LINKAGE SECTION of a program.

Host variables defined in the LINKAGE SECTION of a batch program are often used in different SQL calls in different sub-modules as shown in the following diagram: explains:



Example:

- ID-WS is defined in the WORKING STORAGE section of program A and the linkage sections of both programs B and C.
- Program A calls B. Upon reaching the SQL statement, a call to DSNHADDR establishes addressability on ID-WS. The address is established in the working storage of program A with a value of 123. The SQL statement executes as expected. The SQL-INIT-FLAG is turned off, specifying addressability done.
- Program A calls C. At the start of this program, a new value (456) is moved to ID-WS in preparation for the SQL call.
- Upon reaching the SQL statement, the SQL-INIT-FLAG is off and DSNHADDR is not called. DB2 effectively says it knows the address of ID-WS already and does not need to call DSNHADDR. By not re-establishing addressability, DB2 recognizes the host variable ID-WS as having the value 123, irrespective of what application program C previously moved to it. The SQL statement executes successfully, but returns the same data as application program B (for example, using a key where ID-WS=123, not 456 as expected).

This problem also occurs when a dynamic screen program, running under a drive, requests help that a separate static module services. A request for CA Telon Help in the driver program, therefore, results in a message switch to the static help module. Upon return from the Help screen program, a 04E (invalid data) ABEND occurs in a subsequent SQL statement. Examination of the dump reveals that the address of host variables in the SQL parameter list point to invalid data.

The following series of events occurs:

1. SQL initialization is performed the first time screen A displays. This process establishes the addressability of all host variables used in all SQL statements in the program.
2. A CA Telon Help request in program A results in a message switch to the statically linked help program.
3. When the help program is invoked, the original CA Telon driver program for program A is deleted from memory.
4. Returning from help results in the reloading of the original CA Telon driver in a different area of memory. As a result, the addresses of the variables passed through the linkage section are different from the first display of screen A.
5. The CA Telon driver program calls the original copy of program A, which was not deleted from memory while viewing the Help screen.
6. The driver does not initialize SQL or establish addressability in screen A because SQL-INIT-FLAG equals SQL-INIT-DONE from the previous execution of the program.
7. Subsequent SQL statements reference obsolete host variable addresses until an ABEND occurs.

Note: This is an application problem that often occurs, not a DB2 or CA Telon problem. This problem can also occur in non-CA Telon DB2 programs when two or more higher level modules using different parameter lists call a common module.

Three possible solutions to this problem are:

- Use statically linked drivers containing all the required static modules. This can contradict your existing installation standards and cause application maintenance problems. Also, this solution does not adequately address the situation in which a common module is called from two higher level modules.
- Force SQL initialization (DSNHADDR) to occur every time a screen program or subroutine is called. This is an easy to implement solution for applications where response time is not critical.
- Establish a coding standard that prohibits the use of Linkage Section variables as SQL host variables. This is the preferred solution for both online and batch programs.

BROWSE Request

The browse request was changed to inherit on the ORDER BY parameter and not the KEYCOLS parameter. The KEYCOLS parameter is protected on a browse request. Therefore, to implement a filter portion of the where clause, you must code the complete filter WHERE on the KEY/WHERE parameter. As a final note, the ORDER BY generates both the ORDER BY clause and the paging portion of the WHERE clause. If ORDER BY is updated, the change reflects both the ORDER BY clause and the paging portion of the WHERE.

Plan Name Qualification

This subject discusses the methods of specifying DB2 table and view names on COBOL and PL/I programming, and the limitations imposed by these methods. Ultimately, all table and view names must be fully qualified in a plan used to execute the routine that accesses these tables and views. There are several ways to accomplish this. You can perform the following:

- Code the fully qualified name in the program
- Code an unqualified name, allowing the qualification to be accomplished during bind processing
- Use a synonym, or alternative name, for the tables and views

The following topics discuss each of these methods in more detail.

Fully Qualified Plan Names

To access a DB2 table or view from a COBOL or PL/I program, specify the fully qualified name of the table or view in your SQL statement. The fully qualified name is composed of the owner of the table or view, followed by the table or view name used in the create process. For example, if user JONES creates view EMPLBASE, specify the qualified name JONES.EMPLBASE in the SQL that accesses this view. If user PAYROLL creates a table DEPTDATA, then specify PAYROLL.DEPTDATA in the SQL to access this table.

These examples cause fully qualified names to be present in the DBRM created as output from the DB2 pre-compiler. The bind process then uses this DBRM to create the plan for this application. The CA to this approach is that any authorized user can bind any plan that contains references to these tables and views with the correct results. The major disCA to this approach is that the source code must change whenever a different version of these tables or views is accessed by these routines.

Unqualified Plan Names

An SQL can also reference a table or view from a COBOL or PL/I program by using the UNQUALIFIED name and allowing the qualification to occur at bind. When the bind process encounters an unqualified table reference, DB2 assumes the binder's DB2 authorization ID is the name's qualifier. For example, if JONES executes a bind on a DBRM that contains an SQL access against EMPLBASE, the plan calls for the access against JONES.EMPLBASE.

However, if user PAYROLL performs this same DBRM in a bind, the plan calls for the access against PAYROLL.EMPLBASE. This performs well when the application program accesses only tables and views owned by a single DB2 user, and this user also performs all of the binds. If the routine needs to access the table JONES.EMPLBASE and the table PAYROLL.DEPTDATA, then it is impossible to use unqualified names for all table references.

For JONES to bind a plan that accesses the correct tables/views, the accesses to PAYROLL.DEPTDATA must be fully qualified in the source code. Similarly, for user PAYROLL to bind a plan containing these references, all accesses to JONES.EMPLBASE must be fully qualified in the source code.

DB2 Synonyms

The third approach to SQL referencing of a DB2 table or view in a COBOL or PL/I program uses a SYNONYM. A SYNONYM defines an alternative name for a table or view. Like a table or view name, a SYNONYM name is made up of the owner/creator of the SYNONYM and the synonym name itself. When referencing a synonym in a COBOL or PL/I program, specify the unqualified name. All synonym qualification is complete at bind. At first glance, this coding technique mirrors the unqualified naming technique. The primary difference is that a synonym does not represent DB2 data, it always points at a real table or view containing the DB2 data being accessed. Also, anyone can create a synonym to DB2 data that they are authorized to access.

For our example, all program references to JONES.EMPLDATA and PAYROLL.DEPTDATA are coded as accesses to the unqualified names EMPLOYEE and DEPARTMENT, respectively. The routines are compiled, and the DBRMs created contain references to EMPLOYEE and DEPARTMENT only. User JONES creates the synonyms EMPLOYEE and DEPARTMENT as names for JONES.EMPLDATA and PAYROLL.DEPTDATA, respectively. User PAYROLL also creates synonyms EMPLOYEE and DEPARTMENT as names for JONES.EMPLDATA and PAYROLL.DEPTDATA, respectively.

DB2 actually creates fully qualified synonym names JONES.EMPLOYEE and JONES.DEPARTMENT for user JONES, while PAYROLL creates synonyms PAYROLL.EMPLOYEE and PAYROLL.DEPARTMENT. JONES or PAYROLL can now bind any plan that references EMPLOYEE or DEPARTMENT.

At bind, user JONES causes the plan created to refer to JONES.EMPLOYEE (which points to JONES.EMPLDATA), and JONES.DEPARTMENT (which points to PAYROLL.DEPTDATA). When user PAYROLL binds a plan using DBRM input that references EMPLOYEE and DEPARTMENT, the plan contains references to PAYROLL.EMPLOYEE and PAYROLL.DEPARTMENT; these references in turn reference the actual tables JONES.EMPLDATA and PAYROLL.DEPTDATA. The unqualified synonym names resolve to fully qualified names at bind, and the owner of the SYNONYM is assumed to be the individual performing the bind. When executing routines built using synonyms, execution privileges must be assigned to all users for the plan and the real tables and views involved. You do not grant privileges on a synonym.

If you bind a plan that references a table or view by a synonym, you must create that synonym with your user ID so that the fully qualified synonym name exists when performing the bind. Like table and view names, fully qualified synonym names must be unique. If user JONES creates a table EMPLOYEE, no one can create a view EMPLOYEE nor a synonym EMPLOYEE. However, if JONES creates a table EMPLOYEE and user PAYROLL wants to access this table, PAYROLL can create a synonym EMPLOYEE if PAYROLL has not already created a table, view, or synonym with this name. The synonym name is fully qualified with the owner/creator of the synonym, and it is this fully qualified name that must be unique.

Application Coding Solutions

The following solutions to other common application coding situations are solved using fully qualified table/view names, unqualified table/view names, or SYNONYMs. You often need to execute the same routine against different tables and views. For example, user JONES needs to unit test the program using view TEST.EMPLDATA. This view represents a subset of the table SYSTEST.EMPLDATA accessed from JONES's routine when the application development team is ready to perform system testing.

Using Fully Qualified Plan Names

To accomplish this processing using fully qualified view and table names, perform the following procedure:

1. Code SQL accesses such as `SELECT ... FROM TEST.EMPLDATA`
2. Compile and link-edit, creating a DBRM that references `TEST.EMPLDATA`
3. Bind the DBRM created into a plan for this routine (any authorized user can perform the bind)
4. Perform unit testing
5. Change source SQL accesses to `SELECT ... FROM SYSTEST.EMPLDATA`
6. Compile and link-edit, creating a DBRM that references `SYSTEST.EMPLDATA`
7. Bind the DBRM created into a plan for this routine (any authorized user can perform the bind)
8. Perform system testing

Using Unqualified Plan Names

You can address this same situation using unqualified table and view names, provided that `TEST` and `SYSTEST` are legitimate DB2 userids. This method requires that `TEST` and `SYSTEST` be valid DB2 authorization IDs. If they are not valid, you cannot bind the plans to access the actual view `TEST.EMPLDATA` and the table `SYSTEM.EMPLDATA`. Additionally, this approach implies that no routines in the application need to access a `TEST` view and a `SYSTEST` table from the same program at the same time. You cannot bind a routine that accesses some `TEST` and some `SYSTEST` tables/views using unqualified table/view names.

To use unqualified plan names:

1. Code SQL accesses such as `SELECT .... FROM EMPLDATA`
2. Compile and link-edit, creating a DBRM that references `EMPLDATA`
3. `TEST` bind (performed by a user) the DBRM created into a plan for this routine
4. Perform unit testing
5. `SYSTEST` bind the DBRM created into a plan for this routine
6. Perform system testing

Using Synonyms

Finally, to use synonyms to approach this process, perform the following procedure:

1. Code SQL accesses such as `SELECT .... FROM EMPL`.
2. Compile and link-edit, creating a DBRM that references `EMPLDATA`.
3. Have JONES create a synonym for the view `TEST.EMPLDATA` named `EMPL`. DB2 actually creates a synonym as the fully qualified name `JONES.EMPL`. This fully qualified name must be unique. Therefore, no table, view, or synonym with the fully qualified name `JONES.EMPL` can exist if the create synonym process is performed as indicated. This causes step 6, to require a *drop* if JONES performs it.
4. Have JONES bind the DBRM created into a plan for this routine.
5. Perform unit testing.
6. Either:
 - Have another authorized user create the synonym `EMPL` for the table `SYSTEST.EMPLDATA`, and bind the DBRM previously created into a plan for this routine.
 - Have JONES drop the synonym `EMPL` that references the view `TEST.EMPLDATA`. Then create the synonym `EMPL` for the table `SYSTEST.EMPLDATA`, and bind the DBRM previously created into a plan for this routine.
7. Perform system testing. Having examined how each approach handles the change in tables and views, you can see that you cannot use all approaches in all situations. Fully qualified names require source code changes, while unqualified names require the use of standard IDs for all table and view create and bind processing. You cannot support different owner/creator qualification in a single plan at bind.

Finally, synonyms provide the greatest flexibility. They require each user binding a plan to create a synonym for every unqualified reference found in the DBRM input to the bind.

Naming Conventions

To effectively use SQL, you must carefully plan what you need to do. The naming conventions implemented for DB2 tables, views, and synonyms require extensive planning. The following approach is recommended:

- Define the naming standards for development, system test, and production environments. Present development requirements to demonstrate how you can use the various naming methods to address these requirements.

- Create all production tables and views with a standard DB2 authorization ID. For example, the ID is PROD. A single ID, specified as DBADMIN, creates all production plans. System testing uses another group of tables and views, all created using TEST as the ID. ID TSTADMIN creates system test plans. Unit testing requires that read only processing uses the TEST tables and views, while all update processing requires you to create tables and views. Coding standards require that you code all DB2 accesses using the production synonyms. The production table and view unqualified synonyms are the table and view names themselves.

With these standards in mind, the following chart defines the production and system test tables, views, and synonyms used by a number of application groups in the sample environment.

PRODUCTION:

Tables/views Synonyms

PROD.PAYROLL_EMPLS DBADMIN.PAYROLL_EMPLS
PROD.PAYROLL_SALARY DBADMIN.PAYROLL_SALARY
PROD.PAYROLL_SCALES DBADMIN.PAYROLL_SCALES
PROD.PERSON_EMPLS DBADMIN.PERSON_EMPLS
PROD.PERSON_JOBS DBADMIN.PERSON_JOBS

SYSTEM TEST:

Tables/views Synonyms

TEST.PAYROLL_EMPLS TSTADMIN.PAYROLL_EMPLS
TEST.PAYROLL_SALARY TSTADMIN.PAYROLL_SALARY
TEST.PAYROLL_SCALES TSTADMIN.PAYROLL_SCALES
TEST.PERSON_EMPLS TSTADMIN.PERSON_EMPLS
TEST.PERSON_JOBS TSTADMIN.PERSON_JOBS

These examples follow typical development requirements using these standards.

Example 1

Developer JONES creates a new routine that accesses the production tables PROD.PAYROLL_SALARY and PROD.PERSON_JOBS for read only purposes, and the table PROD.PAYROLL_SCALES for update processing. To unit test this routine, JONES creates the following:

- JONES.PAYROLL_SALARY – a synonym for TEST.PAYROLL_SALARY
- JONES.PERSON_JOBS – a synonym for TEST.PERSON_JOBS
- JONES.PAYROLL_SCALES – a table whose data was copied from TEST.PAYROLL_SCALES

JONES can code accesses using PAYROLL_SALARY, PERSON_JOBS, and PAYROLL_SCALES. At bind, these unqualified references are qualified with the ID JONES, and the correct data is designated in the plan used to unit test the routine. When he/she is ready to start the system testing, JONES drops the table JONES.PAYROLL_SCALES and reuses this name to create a synonym to TEST.PAYROLL_SCALES. JONES then executes a bind and runs the system testing with the newly created plan.

JONES wants to turn the routine over to production. DBADMIN must perform a bind using JONES DBRM and resulting in qualified references to DBADMIN.PAYROLL_SALARY, DBADMIN.PERSON_JOBS, and DBADMIN.PAYROLL_SCALES. This bind requires no changes to the DBRM created at the time the most recent load module was created. This plan is ready for the production environment.

Finally, you must move the unchanged load module into a production load library. Some installations require a compile at this time because only source members are moved into production data sets. This compile requires no source code changes, but the bind must use the DBRM created at this time to insure proper date/time stamp information in the plan for DB2.

Example 2

Developer JONES must modify the routine created as example #1. The modification involves adding a new column to the PROD.PAYROLL_SALARY table for this routine and a number of other new routines being developed. One of these new routines adds the new column's data to the table. You do not need to change all other existing routines that access this table to reference this new column.

Because you cannot use the system test version of the SALARY table, the Corporate Systems Development group creates a unit test version of the SALARY table for JONES and the other developers working on this enhancement project. This table is CRPTST.PAYROLL_SALARY.

To unit test this routine, JONES creates the following:

- JONES.PAYROLL_SALARY – a synonym for CRPTST.PAYROLL_SALARY
- JONES.PERSON_JOBS – a synonym for TEST.PERSON_JOBS
- JONES.PAYROLL_SCALES – a table defined like the PROD.PAYROLL_SCALES table with the new column added

JONES can code accesses using PAYROLL_SALARY, PERSON_JOBS, and PAYROLL_SCALES. At bind, these unqualified references are qualified with the ID JONES, and the correct data is designated in the plan used to unit test the routine. As before, when he/she is ready to start the system testing, JONES drops the table JONES.PAYROLL_SCALES and reuses this name to create a synonym to TEST.PAYROLL_SCALES.

By this time, the definition of the TEST.PAYROLL_SALARY table must be changed to reflect the additional column. JONES drops his/her PAYROLL_SALARY synonym and redefines it as a synonym for the TEST.PAYROLL_SALARY table. JONES then executes a bind and runs the system testing with the newly created plan.

Finally, JONES wants to turn the routine over to production. As with the TEST version of the SCALES table, he/she must redefine the PROD version at this time. DBADMIN can then perform a bind using JONES DBRM that results in qualified references to DBADMIN.PAYROLL_SALARY, DBADMIN.PERSON_JOBS, and DBADMIN.PAYROLL_SCALES. This plan is ready for the production environment. By using synonyms, any authorized user can define DB2 table and view accesses, and establish the appropriate fully qualified DB2 names at bind. This readily allows for changes to the table being accessed and isolates table definition changes to the routines that actually need the changed definitions.

The final issues address what and how to specify this information to the TDF so the CA Telon generator creates the appropriate accesses. For now, CA strongly recommends that you use SYNONYM=Y on all DB2 table/view definitions. This causes CA Telon to generate unqualified accesses in the COBOL or PL/I program that are qualified at bind with the user's ID. Please read the manual on this parameter.

Note: You might want to modify the DB2I module to default the SYNONYM parameter to Y. The DB2I module is the COBOL or PL/I program for the source that provides access to the DB2 catalog during table/view import in TDF option 2.

Executing CA Telon-Developed Applications

To execute a DB2-based application, the CA Telon developed programs must be pre-compiled and bound in the compile/link process:

Pre-compile DB2 Applications

The pre-compiler is a DB2 pre-processor for COBOL, PL/I, and Assembler application programming languages. Its primary function is to remove SQL statements from the source language and replace them with CALL statements. These CALLs indirectly pass control to DB2's Runtime Supervisor when the program is executed. The pre-compiler also constructs a Database Request Module (DBRM) from the SQL statements it encounters, which becomes input to the separate bind process.

A DBRM member for a program is specified through the DBRMMEM symbolic in the generate JCL procedure. The DBRMLIB symbolic specifies the target data set for this member. The values of these two symbolics must be retained to bind an application plan.

Bind DB2 Applications

The bind process compiles together the resultant DBRMs from one or more programs. This produces an application plan that contains the machine code instructions to implement the original SQL requests.

To bind an application plan, use the ISPF BIND/REBIND/FREE option in interactive DB2 using the settings shown below and described in the text following the exhibit.

```

                                BIND
=>

Enter DBRM data set name(s):
 1  LIBRARY(s) ==> 'dbrmLib'
 2  MEMBER(s) ==>  dbmname
 3  PASSWORD(s) ==>

 4  MORE DBRMS? ==> Yes                                (YES to list more DBRMs)

Enter options as desired:
 5  PLAN NAME ..... ==> planname                        (Required to create a plan)
 6  ACTION ON PLAN ..... ==> REPLACE                    (REPLACE or ADD)
 7  RETAIN EXECUTION AUTHORITY ==> Yes                  (YES to retain user list)
 8  ISOLATION LEVEL ..... ==> CS                        (RR or CS)
 9  PLAN VALIDATION TIME ..... ==> RUN                  (RUN or BIND)
10  RESOURCE ACQUISITION TIME ==> USE                   (USE or ALLOCATE)
11  RESOURCE RELEASE TIME .... ==> COMMIT               (COMMIT or DEALLOCATE)
12  EXPLAIN PATH SELECTION ... ==> NO                   (NO or YES)

```

PRESS: ENTER to process END to exit HELP for more information

1. LIBRARY(s) should equal the DBRMLIB value in your generate JCL.
2. MEMBER(s) should equal the DBRMMEM value(s) in your generate JCL and your program name(s). The plural denotes the fact that, when testing multiple DB2 application programs, you must include the DBRMs for each program being tested in one test plan. Also, based on the need for specifying several DBRMs, you might need to get the Extension Panel through the More DBRM's ==> Yes setting.

DBRM's ==> Yes setting. For example:

TRTMT1AD, TRCPT1AD

3. PASSWORD(s) specifies a password if your DBRMLIB is password protected.
4. MORE DBRMS allows you to specify several DBRMs to bind into the application plan.

Granting Public Access on Plans

If the plans you bound did not exist prior to your BIND or a plan was bound with retain execution authority set to 'NO', you become the creator of those plans. To allow use of these plans, (by all students on the CA Telon training, for example), grant public access on these plans using the following SQL under SPUFI:

```
GRANT EXECUTE, BIND
  ON PLAN 'planname'
  TO PUBLIC;
```

Where *planname* = TLNTRGT#(IMS), TTC#(CICS COBOL), or TTP#(CICS PLI)
h2.DB2 Get Diagnostics Support

The following examples show generated code for the Get Diagnostics statement. These statements may be placed either immediately following the data access request (LOCFLAG set to "I") or in a separate G-100 (LOCFLAG set to "G") section.

The code example shown next is generated with the LOCFLAG parameter set to "G", and contains a GDCUST custom code member MYCODE2.

G-100-GET-DIAGNOSTICS SECTION.

```
*****
* G - 1 0 0 - G E T - D I A G N O S T I C S *
*****
* THIS ROUTINE CONTAINS COPY CODE PERFORMED WHEN GET *
* DIAGNOSTICS PARAGRAPHS ARE SPECIFIED. *
* *
* GENERATED - CONTROL LOGIC *
* COPY CODE - OPTIONAL FOR EACH SPECIFICATION *
*****
```

G-100-READNEXT-TRGEMPL-3.

```
EXEC SQL GET DIAGNOSTICS CONDITION :ERR-PTR
      :RSQL-STATE(ERR-PTR) = RETURNED_SQLSTATE
END-EXEC.
```

*TELON-----

*DS: H01 | COPY MYCODE2⁹

*-----

```
IF RSQL-STATE(ERR-PTR) NOT = '00000'
  DISPLAY 'NON-ZERO RETURNED_SQLSTATE = '
      RSQL-STATE(ERR-PTR) ' FOR CONDITION ' ERR-PTR
  ADD 1 NUM-ERRORS GIVING ERR-PTR.
```

*----- | END MYCODE2 ----

G-100-GET-DIAGNOSTICS-RETURN.

The trailing number (for example, 3) in the paragraph name correlates to the sequence number found on the left column of the Get Diagnostics List (S240) screen.

In PL/I, similar code is produced:

```

/*****
* G _ 1 0 0 _ G E T _ D I A G N O S T I C S
*****/

* THIS ROUTINE CONTAINS COPY CODE PERFORMED WHEN GET
* DIAGNOSTICS PARAGRAPHS ARE SPECIFIED.
*
* GENERATED - CONTROL LOGIC
* COPY CODE - OPTIONAL FOR EACH SPECIFICATION
*****/

G_100_READNEXT-TRGEMPL_3: PROC;
    EXEC SQL GET DIAGNOSTICS CONDITION :ERR_PTR
           :RSQL_STATE = RETURNED_SQLSTATE
           ; /* END_EXEC */

/*%INCL C:\TLN50\TLNWORK\MANRI02\109657356.6\MYCODE2.PLI LVL1 */
    IF RSQL_STATE = '00000' THEN DO;
        PUT SKIP EDIT
            ('NON_ZERO RETURNED_SQLSTATE = ', RSQL_STATE,
             ' FOR CONDITION ', ERR_PTR) (A,A,A,P'S9999');
        ERR_PTR = NUM_ERRORS + 1;
        END;
    ELSE
        ERR_PTR = ERR_PTR + 1;
/*END: C:\TLN50\TLNWORK\MANRI02\109657356.6\MYCODE2.PLI */

    END G_100_READNEXT_TRGEMPL_3;

```

The next example shows imbedded Get Diagnostics statements where LOCFLAG is set to "I", and GDCUST custom code member is MYCUST that calls the G-100 section shown in the previous example.

U-100-READNEXT-TRGEMPL.

```

      MOVE 'TRGEMPL ' TO TRACE-SEGMENT-NAME.
      EXEC SQL SELECT  EMPL_ID,
                      EMPL_NAME,
                      EMPL_DOB,
                      EMPL_ZIP
      INTO   :DCLTRGEMPL.EMPL-ID,
            :DCLTRGEMPL.EMPL-NAME,
            :DCLTRGEMPL.EMPL-DOB,
            :DCLTRGEMPL.EMPL-ZIP
      FROM TRGEMPL
      WHERE (EMPL_ID = :XFER-EMPL-ID)
      END-EXEC.
      EXEC SQL GET  DIAGNOSTICS
            :NUM-ERRORS = NUMBER
            :ROW-NUMBER = ROW_NUMBER
      END-EXEC.

```

```

*TELON-----
*DS: H01                                     | COPY MYCUST2
*-----
      MOVE 1 TO ERR-PTR|
      PERFORM G-100-READNEXT-TRGEMPL-3 UNTIL ERR-PTR > NUM-ERRORS. |
*----- | END MYCUST ----

```

Note: The G-100-READNEXT-TRGEMPL-3 paragraph is performed multiple times based on the value of NUMBER retrieved in this Get Diagnostics statement and loaded into host variable NUM-ERRORS.

The code is similar in PL/I as shown following:

```
U_100_READNEXT_TRGEMPL: PROC;

    TRACE_SEGMENT_NAME = 'TRGEMPL ' ;
    EXEC SQL SELECT  EMPL_ID,
                     EMPL_NAME,
                     EMPL_DOB,
                     EMPL_ZIP
    INTO   :DCLTRGEMPL.EMPL-ID,
           :DCLTRGEMPL.EMPL-NAME,
           :DCLTRGEMPL.EMPL-DOB,
           :DCLTRGEMPL.EMPL-ZIP
    FROM TELON.TRGEMPL
    WHERE (EMPL_ID = :WS_EMPL_ID)
    ; /* END_EXEC */

    EXEC SQL GET DIAGNOSTICS
           :NUM-ERRORS = NUMBER
           :ROW-NUMBER = ROW_NUMBER
    ; /* END_EXEC */

/*%INCL C:\TLN50\TLNWORK\MANRI02\109657356.6\MYCUST.PLI LVL1 */
    ERR_PTR = 1;
    DO UNTIL (ERR_PTR > NUM_ERRORS);
        CALL G_100_READNEXT_TRGEMPL_3;
    END;
/*END: C:\TLN50\TLNWORK\MANRI02\109657356.6\MYCUST.PLI */
```

In the COBOL and PL/I examples, the imbedded Get Diagnostics retrieves the number of error conditions for the SELECT statement, and loads that value into a host variable. With that information, a second non-imbedded Get Diagnostics statement that was placed in a G-100 paragraph is then performed in a loop in the custom code member, MYCUST, specified for the first Get Diagnostics statement.

FETCH Details—Using an Alternate Cursor

On the Fetch Details(S244) screen, there are three parameters that impact alternate cursor usage.

- FETCH sensitivity
- FETCH orientation
- ALTCURS

Example:

A situation where there are two data access requests for the same ROW : READNEXT and READLAST. If the READLAST parameter specifies ALTCURS and set to READNEXT_TRGEMPL, then the generated code causes the READLAST data access request to use the cursor defined by the READNEXT data access request. To understand how the ALTCURS feature is used in the generated code, it is instructive to compare the generated code for the READNEXT containing a PRIOR fetch option with that for the READLAST READNEXT containing a LAST fetch option and ALTCURS specification.

The READNEXT with the PRIOR specification generated the following code with four EXEC SQL commands:

```
*****
* DECLARE OF CURSORS PLACED AT START OF PROGRAM          *
*****

* CURSOR: READNEXT_TRGEMPL

      EXEC SQL DECLARE READNEXT_TRGEMPL INSENSITIVE SCROLL CURSOR
      FOR
          SELECT EMPL_ID,
          EMPL_NAME
          FROM TELON.TRGEMPL
          WHERE (EMPL_ID < :EMPL-ID)
          ORDER BY EMPL_ID
      END-EXEC.
*****
* END OF DECLARE CURSORS *
*****
```

Note: All DECLARE CURSOR commands are placed at the beginning of procedural code when an ALTCURS parameter is in use in a program.

In the DECLARE CURSOR statement shown next, the SCROLL token is required for new fetch orientation verbs such as PRIOR.

U-100-READNEXT-TRGEMPL.

```
IF NOT READNEXT-TRGEMPL-CURSOR-OPEN
  PERFORM U-100-READNEXT-TRGEMPL-OPEN.
```

```
EXEC SQL FETCH INSENSITIVE PRIOR
  FROM READNEXT_TRGEMPL
  INTO :DCLTRGEMPL.EMPL-ID,
       :DCLTRGEMPL.EMPL-NAME
  END-EXEC.
```

```
MOVE SQLCODE TO TRGEMPL-STATUS.
```

(generated error processing)

U-100-READNEXT-TRGEMPL-OPEN.

(generated open cursor code)

U-100-READNEXT-TRGEMPL-CLOSE.

(generated close cursor code)

Note: The DECLARE CURSOR, FETCH, OPEN CURSOR, and CLOSE CURSOR calls are generated for this data access request.

However, CA Telon generates much less code for the READLAST READNEXT data access request with ALTCURS specified, since it references a previously defined cursor. As a result, CA Telon generates only a single EXEC SQL as seen in the next example:

U-100-READLAST-TRGEMPL.

```
IF NOT READNEXT-TRGEMPL-CURSOR-OPEN
  PERFORM U-100-READNEXT-TRGEMPL-OPEN.
```

```
EXEC SQL FETCH INSENSITIVE LAST
  FROM READNEXT_TRGEMPL
  INTO :DCLTRGEMPL.EMPL-ID,
       :DCLTRGEMPL.EMPL-NAME
  END-EXEC.
```

```
MOVE SQLCODE TO TRGEMPL-STATUS.
```

(generated error processing)

CA Telon generates the PERFORMs of the cursot open and close, because the processing sequence can not be anticipated.

In the previous example, U-100-READLAST-TRGEMPL, the READNEXT_TRGEMPL cursor is used rather than the non-existent READLAST_TRGEMPL cursor. Since there is no READLAST_TRGEMPL cursor, no READLAST-TRGEMPL-CURSOR-IN variable is generated.

FETCH Details—FETCH for NN Rows

The next example shows the generated coded that CA Telon prduces for the FETCH for NN ROWS option which has been set to 4 in the Fetch Details (S244) screen.

```
EXEC SQL DECLARE READNEXT_MULTROW CURSOR WITH ROWSET

      POSITIONING FOR
            SELECT  EMPL_ID,
                   EMPL_NAME
            FROM TELON. TRGEMPL
      END-EXEC.

      IF NOT READNEXT-MULTROW-CURSOR-OPEN
        PERFORM U-100-READNEXT-MULTROW-OPEN.
      MOVE 'MULTROW ' TO TRACE-SEGMENT-NAME.
      MOVE SPACES TO TRACE-FIELD-NAME.
      EXEC SQL FETCH NEXT ROWSET
            FROM READNEXT_MULTROW
            FOR 4 ROWS
            INTO :TESTID,
                :TESTNAME
      END-EXEC.
```

The receiving host variables in this example must be defined as follows using OCCURS 4, since we have specified a fetch for 4 rows:

```
01 TESTID-TABLE.
   05 TESTID          PIC X(6) OCCURS 4.
   05 TESTNAME        PIC X(25) OCCURS 4.
```

User DATATYPES

When user datatypes are specified for SENCOLS columns in the Update User-Defined Datatypes (S184) screen, they are generated as CAST...AS clauses in a SELECT statement.

Example:

If the following user datatypes are specified in screen S184:

SENCOLS Column	Datatype
EMPL_DOB	CHAR (6)
EMPL_ZIP	NUMERIC

CA Telon Generator produces the following code:

```
EXEC SQL DECLARE TRANSACT_TESTROW CURSOR FOR
      SELECT EMPL_ID,
             EMPL_NAME,
             CAST (EMPL_DOB AS CHAR(6)),
             CAST (EMPL_ZIP AS NUMERIC)
      FROM TELON.TRGEMPL
      ORDER BY EMPL_ID
      END-EXEC.
```

Declaring Global Temporary Tables

Specification of DB2 Declare Global Temporary Tables is dealt with by two TDF screens, Create/Update SQL Table (D141) and Select New Row Name (S137). Any temporary tables defined within a program will have their name, qualifier, commit options, and custom code defined at the ROW level. Therefore, ROW may be designated as corresponding to a temporary table. A sample of generated code that might be produced to support global temporary tables is shown in the next example.

The TMPNAME parameter indicates that a DECLARE TEMPORARY GLOBAL TABLE statement is generated with the name specified in this parameter (in this case, MTTMPTBL). The TMPCOMT parameter causes the generation of an ON COMMIT phrase (SAVE which causes a PRESERVE ROWS clause to be generated. The column names and data types arise from the usual DCLCOL settings.

With the above specifications, CA Telon generates the following code for the temporary table:

```
PROCEDURE DIVISION.  
  EXEC SQL DECLARE GLOBAL TEMPORARY TABLE SESSION.MYTMPTBL  
    (EMPL_ID CHAR(6) NOT NULL,  
     EMPL_NAME CHAR(25) NOT NULL,  
     EMPL_ZIP CHAR(5) NOT NULL)  
    ON COMMIT PRESERVE ROWS  
  END-EXEC.
```

Note: The qualifier, SESSION, is generated as a default, if the TMPQUAL parameter is not specified.

Appendix E: Using The DL/I Environment

This appendix addresses issues that relate to IBM's IMS/DB (DL/I) database management system.

This section contains the following topics:

[BOOLEAN SSA and Secondary Indexes](#) (see page 637)

[Using Multiple PCBs for One Database](#) (see page 644)

BOOLEAN SSA and Secondary Indexes

This subject discusses the use of the CA Telon Design Facility for specifying alternative SSAs, and for accessing DL/I segments by secondary index paths. This subject assumes basic knowledge of TDF Option 2 (Data Administration) DBD and PSB specification, and Option 4 or 5 (Data Group processing).

CA Telon option 2 processing automatically defines an SSA for every keyed segment of a DL/I database defined to it. This SSA always has the same structure:

*segname *---(primary-key = xxx...)*

- *segname*—The segment name
- *primary key*—The primary IMS key defined in the DBD
- *xxx*—The area filled in by data defining the key value processed

This SSA is generated and used for all default DL/I accesses to the specified segment defined in your program definition's Data Group. Within option 2, the definition of this SSA is referenced as the ****DFLT**** DLIDSC, where DLIDSC indicates a DLI or EXEC DLI Data Search Criteria, and ****DFLT**** refers to this entry as being the default DLIDSC.

When a database segment has a secondary index associated with it, CA Telon defines another DLIDSC that can generate a second SSA for that segment. This DLIDSC is referred to by the name of the secondary index, the PROCSEQ, for DL/I processing. The secondary index SSA generated from this DLIDSC is structured exactly like the primary default SSA. The name of the secondary index key replaces the IMS primary key value referenced by primary-key in the above SSA structure. You can then define a PSB or File Group in option 2 that contains this database with the secondary index as the PROCSEQ. Any DL/I accesses defined in your program definition for this database automatically use the secondary index SSA.

CA Telon generated applications can also use other SSAs for accessing databases. Typically, applications require SSAs that reference search keys or require BOOLEAN processing of the primary or secondary keys. For these reasons, you can construct alternative SSA structures for any keyed segment in TDF option 2 by defining other DLIDSCs. The following description creates and accesses alternative SSAs within the CA Telon Design Facility using a database that contains three keyed segments, the first of which also has a secondary index.

Request an update of the DL/I database from the Option 2 menu. This presents a display of the database's DBD information.

Note: The DBD involved is EDSTUDBD, as shown in the upper-left corner. The segments defined to this database are EDSTUDN1, EDSTUADR, and EDSTUCLS, as shown on lines 2, 5, and 6 of this list.

The primary IMS key for the EDSTUDN1 segment is EDSTUKEY, a nine-character SSN defined on line 2. Also, EDNAMX is defined as a logical child of the EDSTUDN1 segment. This LCHILD references a secondary index DBD, EDNAMIDX. The secondary index key is a 30-character name, EDNAME, as noted on line 4 of the list.

Request an update of the segment that needs alternative SSAs by placing a **U** on the sequence line for that segment.

```
EDSTUDBD UPDATE DBD *****
COMMAND ==> _____ PAGE 01

ACCESS HIDAM.VSAM RMNAME

SEQ TYPE NAME PARENT/DEVICE MAX LTH SEGMENT KEY LENGTH START
001 DATA SET EDSTUDBD 3350 _____
U 2 SEGM_ EDSTUDN1 0 _____ 150 EDSTUKEY 9 1
003 LCHILD_ EDIMDX_ EDSTUIDX_ _____
004 LCHILD_ EDNAMX_ EDNAMIDX_ _____ EDNAME 30
005 SEGM_ EDSTUADR EDSTUDNT 150 EDADRKEY 4 1
006 SEGM_ EDSTUCLS EDSTUDNT 150 EDCLSKEY 5 1
```

This screen, an example of the DBD segment update, shows the default primary DLIDSC.

```
UPDATE DBD: EDSTUDBD SEGM: EDSTUDNT *****
COMMAND ==> _____
OPTIONS ==> SEARCH FIELDS + PCB PARMS _ DLIDSC x

GENERAL: LABEL _____
* COPY _____ COPYLV1 _ COPYLBL _____

** DLIDSC SEGMENT CMND IMSKEY OP KEY
01 **DFLT** EDSTUDNT * _ EDSTUKEY = _
```

To reference other DLIDSCs that can generate alternative SSAs in your COBOL or PL/I program, enter **X** next to DLIDSC on line 3 of the screen. The DLIDSC update panel displays:

```

EDIT ____ UPDATE DLIDSCS FOR SEGMENT MEMBER 001 OF 001 SIZE 000002 COL 01
COMMAND ==> SCROLL ==> CSR
DBD: EDSTUDBD SEGMENT: EDSTUDNT

***** DLIDSC USECNT CMMD IMSKEY OP KEY MORE
***** ***** ***** ***** ***** ** ***** *
000001 **DFLT** * ____ EDSTUKEY =
000002 EDNAMIDX * ____ EDNAME = X
***** ***** ***** ***** ***** ** ***** *
    
```

The DLIDSC update screen displays for the EDSTUDN1 segment updated. The first DLIDSC is the default shown on the DBD Segment Update Panel, and the secondary index DLIDSC is listed and named EDNAMIDX. The second is the DBD name referenced by the LCHILD statement defining this index in the previous update DBD panel. The EDNAMIDX DLIDSC has an **X** in the far-right column indicating this is a secondary index DLIDSC.

```

EDIT ____ UPDATE DLIDSCS FOR SEGMENT MEMBER 001 OF 001 SIZE 000003 COL 01
COMMAND ==> SCROLL ==> CSR
DBD: EDSTUDBD SEGMENT: EDSTUDNT

***** DLIDSC USECNT CMMD IMSKEY OP KEY MORE
***** ***** ***** ***** ***** ** ***** *
000001 **DFLT** * ____ EDSTUKEY *
000002 EDNAMIDX * ____ EDNAME * X
u 0003 ednamI2b * ____ EDNAME * X
***** ***** ***** ***** ***** ** ***** *
    
```

You can create additional DLIDSCs that reference the primary IMS key or search fields using the standard line editor commands: insert(I), copy(C), or repeat(R). To create DLIDSCs that reference a secondary index, you *must* copy(C) or repeat(R) the secondary index DLIDSC provided by CA Telon (or one previously created from it). Type over the DLIDSC name field to create a unique name for the new DLIDSC. Use the DLIDSC update panel to copy the EDNAMIDX DLIDSC and name it **EDNAMI2B**.

Press Enter and the Update SSA/Command Panel appears.

```
UPDATE SSA/CMD FOR DL/I SEGMENT * *****  
CMD ==>  
DBD: EDSTUDBD SEGMENT: EDSTUDNT DLIIDSC: EDNAMI2B PROCSEQ: EDNAMIDX  
  
GENERAL: KEYFEED _____  
* CMDCODE * ____ -OR- PATH_ (Y/N) CURRENT _ (Y/N) OPTION _ (F/L)  
* CONCATK ____ (Y/N) PARENTG_ (Y/N) LOCKED _ (Y/N)  
EX DL/I: VARLTH _ (Y/N) OFFSET ____  
  
WHERE/BOOLEAN SSA: BOOL  
U IMSKEY OP KEY OF  
_ EDNAME_ = _____  
_____  
_____  
_____  
_____  
_____  
_____  
_____
```

You can update EDNAMI2B to modify its specifications. For example, to alter it to generate a BOOLEAN SSA, enter a **U** in the sequence number of the DLIDSC. Press Enter to display the DLIDSC detail update panel for the EDNAMI2B DLIDSC created. Enter BOOLEAN information.

```
UPDATE SSA/COMMAND FOR DL/I SEGMENT * *****  
COMMAND ==> _____  
DBD: EDSTUDBD SEGMENT: EDSTUDNT DLIDSC: EDNAMI2B PROCSEQ: EDNAMIDX  
  
GENERAL: KEYFEED _____  
* CMDCODE * ____ -OR- PATH _ (Y/N) CURRENT _ (Y/N) OPTION _ (F/L)  
* CONCATK _ (Y/N) PARENTG _ (Y/N) LOCKED _ (Y/N)  
EX DL/I: VARLTH _ (Y/N) OFFSET ____  
  
WHERE/BOOLEAN SSA: BOOL  
U IMSKEY OF KEY OF  
_ EDNAME _ > xfer-start-name_____ and  
_ edname _ < xfer-end-name_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____
```


Perform normal END processing until the Option 2 menu displays. This saves the DLIDSC information you created and associated with the EDSTUDBD database. If you are unsure about the search key names and IMS keys available, you can request a list by entering a **U** in the column to the left of the IMSKEY name columns.

```

TRTEST.SD AUTOEXEC OUTREAD EDSTUDM1 *****
COMMAND ==> _____
OPTIONS ==> PREVIEW _

GENERAL: FUNC _____ SSALIST _ (Y/N) CONCATK _ (Y/N)
* IGNORE _____
* IOAREA @ _____
* PATH _____ PARENTG _____ LOCKED _____ CURRENT _____
** DLIDSC SEGMENT CMMD IMSKEY OP KEY
01 **DFLT** EDSTUDN1 @* _____ @EDNAME @= _ @? _____

```

To use the newly created DLIDSC in an application program, you must bring the PSB created with this database specification and the PROCSEQ of EDNAMIDX (the secondary index) into your data group. Request an AUTOEXEC or a USEREXEC IO detail update panel to select EDNAMI2B as the DLIDSC used for the generated DL/I database access. On initial access, the standard secondary index DLIDSC is referenced because it is the default for an access of a segment with a secondary index PROCSEQ attached to it.

From the detail IO update screen, you can enter the appropriate DLIDSC name if you know it, or you can enter a **U** on the sequence number to view the DLIDSC list from TDF Option 2. This is the same list presented in the updated DLIDSC update panel shown earlier. Enter **S** on the sequence number of one of the DLIDSCs presented to SELECT it as the DLIDSC used for this DL/I access.

Because this is the Option 2 DLIDSC list panel, you can also perform Option 2 functions from this list, provided that database's Option 2 data is not being updated at the same time. You can also create a BOOLEAN DLIDSC directly from Option 4 or 5 data group processing using the detail IO update panel.

```

EDIT _____ UPDATE DLIDSCS FOR SEGMENT MEMBER 001 OF 001 SIZE 000003 COL 01
COMMAND ==> SCROLL ==> CSR
DBD: EDSTUDBD SEGMENT: EDSTUDN1

***** DLIDSC USECNT CMMD IMSKEY OP KEY MORE
***** ***** ***** ***** ***** ***** ***** ***** *****
000001 **DFLT** * _____ EDSTUKEY =
000002 EDNAMIDX * _____ EDNAME = X
S 0003 EDNAMI2B * _____ EDNAME > XFER-START-KEY X
***** ***** ***** ***** ***** ***** ***** ***** *****

```

Press Enter; an updated panel appears.

```

TRTEST.SD AUTOEXEC OUTREAD EDSTUDN1 *****
COMMAND ==> _____
OPTIONS ==> PREVIEW _

GENERAL: FUNC _____ SSALIST _ (Y/N) CONCATK _ (Y/N)
* IGNORE _____
* IOAREA @ _____
* PATH _____ PARENTG _____ LOCKED _____ CURRENT _____
** DLIDSC SEGMENT CMMD IMSKEY OP KEY
01 EDNAMI2B EDSTUDN1 @* _____ @EDNAME @>_ @XFER-START-NAME_____
    
```

In either case, the detail IO update panel contains a reference to EDNAMI2B as the DLIDSC for this DLI database access.

The generator creates an SSA for each DLIDSC in your screen or batch program. Each access is generated with the appropriate SSA and moves the appropriate program variables to qualify those SSAs. The previous sample generates the following COBOL code:

```
*
* EDSTUDN1-EDNAMI2B-SSA
*
05 EDSTUDN1-EDNAMI2B-SSA
10 EDSTUDN1-EDNAMI2B-SSA-SEGMENT PIC X(8)
VALUE 'EDSTUDN1'.
10 EDSTUDN1-EDNAMI2B-SSA-CMDCODE PIC X(4)
VALUE '*___'.
10 EDSTUDN1-EDNAMI2B-SSA-LPAREN PIC X(1)
VALUE '('.
10 EDSTUDN1-EDNAMI2B-SSA-IMSKEY1 PIC X(8)*
VALUE 'EDNAME'.
10 EDSTUDN1-EDNAMI2B-SSA-OPCODE1 PIC X(2)
VALUE '>'.
10 EDSTUDN1-EDNAMI2B-SSA-VALUE1 PIC X(30).
10 EDSTUDN1-EDNAMI2B-SSA-BOOLOP1 PIC X(1)
VALUE '&'.
10 EDSTUDN1-EDNAMI2B-SSA-IMSKEY2 PIC X(8)
VALUE 'EDNAME'.
10 EDSTUDN1-EDNAMI2B-SSA-OPCODE2 PIC X(2)
VALUE '<'.
10 EDSTUDN1-EDNAMI2B-SSA-VALUE2 PIC X(30).
10 FILLER PIC X(2)
VALUE ')'.
:
:

A-100-OUTPUT-INIT SECTION.
:
:
MOVE XFER-START-NAME TO
EDSTUDN1-EDNAMI2B-SSA-VALUE1.
MOVE XFER-END-NAME TO
EDSTUDN1-EDNAMI2B-SSA-VALUE2.
CALL 'CBLTDLI' USING GU-FUNC
STUDENT2-PCB
IOA-EDSTUDN1-SEGMENT
EDSTUDN1-EDNAMI2B-SSA
:
:
```

Using Multiple PCBs for One Database

Create the PSB as desired and import it. Use Option 2 of the TDF (Data Administration) to update the PSB. Rename the second PCB to make it unique. Update hhPROC and hhPCBS, if necessary.

On the DG screen (Option 4) for the second PCB:

- Change the labels on the segments (you can enter this just once in Option 2 and all PCB users inherit it)
- Put COPY=**NONE** for each segment, if using the same IOAREA as the calls in the first PCB

When you use the second PCB, these steps use the IOAREA from the first PCB, but use the SSAs for the second.

Note: If you added the second PCB later, it might be easier to create a new PSB for the program involved, rather than making a change that would affect other programs.

Appendix F: Importing Data Inheritance

This appendix describes the process by which CA Telon definitions, DL/I databases, and PSBs are brought into a TDF. Importing is the CA Telon facility that brings these items into a TDF. Of special importance is the impact this process can have on definitions already in the TDF related to their potential for changing inherited data. Import provides processing to maintain the functional integrity of these TDF definitions.

Inherited data is information that is shared by multiple CA Telon program definitions and is stored in a single location. TDF Option 2, Data Administration, currently contains all information that can be in CA Telon definitions.

When you import CA Telon program definitions and Database Definitions (DBDs), CA Telon importing produces a report showing active TDF data administration information that is different from the information contained in the CA Telon definition or DBD source. You can then allow CA Telon to resolve the differences or manually resolve them yourself.

This appendix covers the following subjects:

- **Overall Approach and Recommendations**—Shows how and when to use CA Telon Import facilities, describes basic terminology for import processing, gives recommendations for import processing, and defines the relationship between import processing and inherited data
CA Telon Definition Importing—Shows how to import CA Telon definitions from PDSs or CA Panvalet libraries and describes all parameters, reports, jobs, procedures, and messages relevant for CA Telon definition imports
- **DL/I DBD Importing**—Shows how to import DL/I Database Definitions (DBDs) from PDSs and describes all parameters, jobs, procedures, and messages relevant for DL/I DBD imports
- **DL/I PSB Definition Importing**—Shows how to import DL/I Program Specification Block (PSB) Definitions from PDSs and describes all parameters, jobs, and procedures relevant for DL/I PSB Definition imports

Overall Approach and Recommendations

This subject presents the proper approach to importing, describes basic import terminology and gives recommendations for import processing.

Proper Application Storage

CA Telon-generated production applications are typically stored in PDSs or CA Panvalet libraries rather than the TDF VSAM files from which they were created. PDSs and CA Panvalet provide better version control, backup, and security processing. Consequently, you may often need to bring an application definition into a TDF environment from one of these storage media. To do this, use CA Telon definition import processing.

Additionally, DL/I database information used by CA Telon-generated applications can also be brought into a TDF environment directly from the DL/I DBD and PSB source used to create the databases and PSBs used by the applications. Use CA Telon DBD and PSB imports to accomplish this. How you use CA Telon's importing facilities is highly dependent upon your installation's CA Telon procedures, standards, and your TDF environment(s).

Typical Import Usage

Import processing is typically required when a CA Telon-generated application is undergoing change. This change is most often the result of either:

- Production problems which require fixes/maintenance
- Application development efforts

The system's development effort has been divided into these categories because of their relationships with the data definitions they use. Production fixes/maintenance typically involve stable data definitions, while application development is often using data whose physical and even logical structure is subject to change.

Factors Influencing Data Definition Stability

The stability of the data definitions directly impacts the required scope of the CA Telon application definition import. If the Data Administration data is stable, minimal discrepancies can be expected. On the other hand, when the data definitions are undergoing frequent correction and enhancement, the older a CA Telon definition is, the more likely it is to be different from the TDF Option 2 data definitions.

Implied in the above statements is the existence of data definitions against which to compare the imported source. You may in fact be using a TDF that does not contain Option 2 data definitions that match the data items referenced by the importing CA Telon definition. This also impacts how import processing should be performed.

Input Parameters

CA Telon import processing uses two input parameters to control Data Administration updates. The first parameter is the RUNTYPE parameter, which controls how the imported data is interrelated with the corresponding TDF data (compared, overlayed, merged, or ignored). The second parameter is the MAXSEVR parameter, which determines the maximum severity allowed for an import run.

RUNTYPE parameter

There are four values for the RUNTYPE parameter: **C**, **O**, **M**, and **I**.

C (COMPARE)

Compares the imported source's data administration information with its corresponding TDF Option 2 data and reports the differences (**C** is not valid for PSB import processing).

Note: CA Telon definition import processing may also update TDF Option 4 or 5 with the CA Telon definition, but never updates TDF Option 2 information.

O (OVERLAY)

Replaces the DL/I DBD or PSB data contained in TDF Option 2 with the importing source data (**O** is not valid for CA Telon definition importing). During DBD import processing, comparison of importing source data against TDF Option 2 data is performed and the results reported.

M (MERGE)

Performs the same comparison and reporting processing as RUNTYPEs **C** and **O** while potentially updating the TDF Option 2 data associated with the CA Telon definition or DBD being imported (**M** is not valid for PSB importing). RUNTYPEs **M** and **O** specific differences. Each difference encountered by the comparison processing is assigned a severity rating. RUNTYPE **M** ratings.

I (IGNORE)

Adds imported data to the TDF, ignores the TDF option 2 data, and performs no comparison during import. Although importing **I** is valid for all items, ignore PSB and DBD import runs require that the TDF does not already contain definitions for the importing item(s).

MAXSEVR Parameter

Set the MAXSEVR parameter at import run time to determine the maximum severity allowed before import processing is aborted. When comparison processing is activated, each comparison creates a message in a file that is input to the import report routine. Each message has a severity code assigned to it. The severity codes are compared against the MAXSEVR parameter.

If any message created for an import has a severity exceeding the maximum allowable severity that you set for the MAXSEVR parameter, the definition is not imported. When a run exceeds the maximum severity, Data Administration information is not changed, even if the run type is reque Administration Data should be merged or overlaid. In all cases, regardless of the MAXSEVR value, a severity **16** message will cause the import to abort.

Messages

Every import run that makes comparisons of data (run types **C**, **O** and **M**) creates a message in a file for each difference found. This file is input to the import report routine. All messages are shown later in this appendix under the subject "Messages".

Severity Codes

Each message has a severity code assigned to it. As described above, the severity codes are compared against the MAXSEVR parameter to determine whether or not importing will occur.

Possible severity code values are 0, 2, 4, 8, 12, and 16. Zero (0) indicates either no differences or no item on the TDF for comparison, and sixteen (16) indicates major structural differences (which will cause a severe error). The numbers in between show varying degrees of difference.

Severity codes are described more fully later in the appendix under the subject "Severity Codes".

Reports

Every import run produces a report, no matter which run type is used. However, all PSB importing as well as all importing using run type **I** (IGNORE) produce only import status messages with no Data Administration comparison messages. When you run a DBD or CA Telon definition import with run type **C**, **O**, or **M**, you produce a report showing all discrepancies between the TDF definitions and the import data definitions. A sample report is shown later in this appendix under the subject "Sample Report".

Recommendations for Import Processing

The following information defines various ways to use CA Telon definition and DBD import processing both for production maintenance and new development. The description of import processing is broken down of development effort, and secondly, by the scope of the TDF Data Administration contents (if this is a factor).

Production TDF Available

This category typically uses definition import with a run type of MERGE and a maximum severity of **4**. The definitions generally import successfully as database/data set definition changes are not expected. With a maximum severity of **4**, CA Telon-related data in the definition being imported can be added to the TDF Option 2 data.

However, database/data set inherent data cannot be inadvertently changed by import. Specifically, DL/I DSCs (SSAs) defined in the importing definition are added to TDF Option 2 if they do not already exist. Similarly, data identified as inherited from TDF Option 2 loses its inheritance indicator if it no longer matches the Option 2 values. All other changes are flagged with a severity greater than **4**, causing no import to occur.

When a discrepancy with a severity greater than **4** aborts the import, you must analyze the discrepancies as presented by the definition report to determine their impact on the definition being imported or the data definitions. You can then re-import after:

- Upping the maximum severity AND/OR
- Changing the TDF Option 2 data AND/OR
- Running with an IGNORE import run type

Note: Running with an IGNORE run type forces the definition import with NO changes to Option 2. You are responsible for all changes to Option 2 and/or Option 4 to resolve the discrepancies. Upping the severity brings the definition in but the Option 2 data is modified as is indicated in the import report. These changes should be reviewed for appropriateness.

The highest recommended maximum severity is **12**.

Empty TDF Available

This category typically uses definition import with a run type of MERGE and a maximum severity of 12. Since the TDF into which you are importing is empty, the expected result is a series of *ADD* processes.

Note: This approach requires that you delete Option 2 data each time you complete your changes, since the next import expects Option 2 to be empty. This implies that all fix processing be *single-threaded*.

Any time you are maintaining multiple definitions that access any or all of the same data items, the first definition imported adds the data definitions. Consequently, all subsequent definitions are not importing into an empty TDF. Furthermore, if this is a DL/I environment, all DBDs referenced *first* should be imported via DBD source import and the Production TDF approach should be used instead. This will preclude the *single-thread* requirement.

Application Development

The development category typically involves Data Administration data that is undergoing development (for example, it is changing). In this environment, you can expect discrepancies between the importing definitions' views of data and the TDF's view of that data. It is also likely that you will want any imported definition to be brought into sync with the TDF's view.

Keeping these issues in mind, you can best resolve non-program-specific discrepancies manually, after the definition itself has been imported. Thus, definition import should use **C** (COMPARE) with a maximum severity of **12**. In this way, all discrepancies are reported so that you can determine what changes to make to TDF Option 2 and/or Option 4 data. The definition is also imported in all cases except when a DL/I database has undergone a hierarchical change that invalidates I/O.

Importing an Object

Importing an object (PI, SD, etc.) into the TDF when that object already exists and is being edited at the time of import may cause unpredictable results.

CA Telon Definition Importing

CA Telon import processing provides an automated facility for entering CA Telon source definitions into the TDF. The imported definition can be a CA Telon SD, BD, RD, or DR.

Processing

The CA Telon import procedure *optionally* compares/updates and reports on DBMS with differences between CA Telon source and TDF Option 2 (Data Administration) data. Compare/update has three distinct run types that apply to CA Telon definition importing: **C**, **M**, and **I**.

Each run type imports a CA Telon definition if serious errors are not encountered. In addition, run types **C**, **M**, and **I** are able to report on and/or update the TDF Data Administration data that is related to the source being imported.

Note: Definition imports that perform DB2 comparison processing can erroneously report a **TLNROW NOT FOUND** message. Access this table for update processing from the TDF Option 2 menu, and exit without making any changes. After doing this, retry the import and the error should not occur.

Run Type C (Compare)

Run type **C** results in a *comparison* of the CA Telon source Data Administration data against the TDF Data Administration. Each difference is reported and rated using a severity code of **0**, **2**, **4**, **8**, **12**, or **16**. The TDF Data Administration data do not occur. The CA Telon source, however, is imported, provided the highest severity code is less than or equal to the maximum allowable severity code.

For example, using run type **C** with a maximum severity code of **0** results in an import of the definition only if there were *no* differences. On the other hand, using run type **C** with a maximum severity code of **16** *always* results in an import of the definition.

The following table summarizes the TDF updating associated with COMPARE type imports.

Allowed	Encountered	Imported?	Updated?
0	0	Y	N
	> 0	N	N
2	< = 2	Y	N
	> 2	N	N
4	< = 4	Y	N
	> 4	N	N
8	< = 8	Y	N
	> 8	N	N

Allowed	Encountered	Imported?	Updated?
12	< = 12	Y	N
	> 12	N	N
16	< 16	Y	N
	= 16	N	N

Run Type M (Merge)

Run type **M** operates like run type **C** with the added ability to update the TDF Data Administration data. Like run type **C**, differences between the CA Telon definition being imported and the TDF Data Administration data are rated by a severity code. The Data Administration data has values changed to the importing definition's values when there are differences provided that:

- Differences are not rated higher than the maximum allowable severity set by the maximum severity parameter (MAXSEVR) and
- The current Data Administration data is blank (for example, CA Telon Definition data is being *added* to the TDF Data Administration data). Current TDF Data Administration values are not being changed. When CA Telon Definition and Data Administration values differ, inheritance is removed from the definition (if possible), localizing the values to that definition *only*.

Like check processing, the CA Telon definition is imported if the maximum allowable severity is not exceeded.

The following table summarizes the TDF updating associated with MERGE type imports.

Allowed	Encountered	Imported?	Updated?
0	0	Y	Y
	> 0	N	N
2	< = 2	Y	Y
	> 2	N	N
4	< = 4	Y	Y
	> 4	N	N
8	< = 8	Y	Y
	> 8	N	N
12	< = 12	Y	Y
	> 12	N	N
16	< 16	Y	Y
	= 16	N	N

Ideally, MERGE runs are limited to when either the production TDF has been changed or a CA Telon definition is being imported into a TDF that did not previously contain the definition. With this in mind, the most frequently encountered differences are in the values of inherited data or in the DLIDSCs defined.

Note: If any of the data administration data referenced in the importing definition is in use by another import run or by a TDF user, the import/MERGE does not take place.

Severity code 0

If no differences are encountered, a severity of **0** is assigned. A severity **0** is also assigned when a database or data set reference in the importing source is not defined in the TDF data administration. This results in the addition of these databases and/or data sets.

Severity codes 2, 4, and 8

If inherited values are different on the TDF, the imported source values are designated as non-inherited values and assigned a severity code of **2**.

If DLIDSCs are used by the definition and are not defined to the TDF and if the maximum severity code is at least **4**, the DLIDSC are added to the TDF and assigned a severity code of **4**.

When a database component (such as, DL/I segment or DB2 TLNROW) used in the definition does not exist on the TDF and the maximum severity code is at least **8**, a severity code of **8** is assigned and the database component is added. Similarly, when TDF values are blank and source data definition parameters are not, a severity code of **8** is assigned to these comparison messages. If the maximum severity code is an **8**, the source values are added to the TDF.

Therefore, if the differences are rated with a severity code of **4** or less, the TDF Data Administration data can probably be updated, using the referenced databases and/or data sets, without impacting other development. This is because changes are confined to the CA Telon-used application data.

If you want to minimize the automatic changes made to TDF Data Administration based on imported application definitions, a MERGE run type is best used with an allowable severity code of **4**. Using a maximum severity code of **8** may result in some changes to the Data Administration data inherent to the databases and data sets defined there; however, these changes are additions that may not impact other application definitions using them.

Severity codes 12 and 16

If you are not concerned with Data Administration changes, a MAXSEVR parameter value of **12** with a run type of **M** almost ensure that your definition is imported and your Data Administration is in sync with your application source. A maximum severity of **16** is never recommended with run type **M**, as severe inconsistencies could exist between the TDF data and the importing source data definitions.

Note: Definition importing with a run type of **M** does not delete DL/I database segments; however, it may cause segments to be added to the DL/I database. Consequently, DL/I databases that have segments added as a result of input processing should be reviewed for structural integrity.

Run Type I (Ignore)

Run type **I** provides for import processing that IGNORES the Data Administration data contained in the TDF. This implies that there is no compare or update processing of the TDF Data Administration data. No comparison means no severity codes, so the CA Telon definition always imports. This import type can be used whenever TDF Data Administration changes are not a concern. The following table summarizes the TDF updating associated with IGNORE type imports.

Allowed	Encountered	Imported?	Updated?
0	N/A	Y	N/A
2	N/A	Y	N/A
4	N/A	Y	N/A
8	N/A	Y	N/A
12	N/A	Y	N/A
16	N/A	Y	N/A
20	N/A	Y	N/A

Summary

The following table summarizes the processing to be performed with CA Telon definition importing.

Runtype	No Severe Error Encountered	Severe Error Encountered
C-COMPARE	■ Data Administration data compared and differences	■ Data Administration data compared and

Runtype	No Severe Error Encountered	Severe Error Encountered
	- reported ■ CA Telon definition imported ■ Report of definition import results produced	- differences reported ■ CA Telon definition NOT imported ■ Report of definition imported
M-MERGE	■ Data Administration data compared and differences reported ■ Data Administration data added to if TDF data is blank and source is not ■ CA Telon definition imported ■ Report of definition import results produced	■ Data Administration data compared and differences reported ■ Data Administration data NOT changed ■ CA Telon definition NOT imported ■ Report of definition import results produced
I-IGNORE	■ Data Administration data NOT compared ■ Data Administration data NOT updated ■ CA Telon definition imported ■ Report of definition import results produced	NOT APPLICABLE

Importing from a PDS

Run an OS/390 or z/OS Batch job to import a CA Telon definition. The model JCL provided in the INSTALL data set is JUMDEF, which executes procedure TLNUMDEF. The CA Telon source for this module must be in a PDS member.

The most frequently used input parameters to JUMDEF, as defined in TLNUMDEF, are as follows:

RUNTYPE

Import run type:

C

Compare only

M

Import and merge TDF

I

Import (the default) – do not compare or change the TDF data administration

MAXSEVR

Maximum allowable message severity for import to complete. The default is **0**.

SRCMEM

CA Telon definition source member name in the source library (see SRCLIB).

SRCLIB

CA Telon definition source PDS.

VSQUAL

High level data set name qualifier for the TDF VSAM files into which the definition will be imported.

Importing from CA Panvalet

Importing a CA Telon definition from an CA Panvalet library is accomplished by running the batch job stream JUMPAN, which in turn executes procedure TLNUMPAN. The most frequently used input parameters to TLNUMPAN are as follows:

RUNTYPE

Import run type:

C

Compare only

M

Import and merge TDF

I

Import (the default) – do not compare or change the TDF data administration

MAXSEVR

Maximum allowable message severity for import to complete. The default is **0**.

SRCMEM

CA Telon definition source member name in the CA Panvalet library (see PANLIB)

PANLIB

CA Telon definition source CA Panvalet library.

VSQUAL

High level data set name qualifier for the TDF VSAM files into which the definition will be imported.

Importing from AllFusion Endeavor Change Manager

Importing a CA Telon from a CA Software Change Manager for Mainframe library is accomplished by running the batch job screen JUMDFE, which in turn excutes procedure TLNUMDFE are as follows. The most frequently used input parameters to TLNUMDFE are as follows:

RUNTYPE

Import run type:Import (the default) – do not compare or change the TDF data administration Import run type:

C

Compare only

M

Import and merge TDF

I

Import (the default) - do not compare or change the TDF data administration

MAXSEVR

Maximum allowance message severity for import to complete. The default is 0.

SRCMEM

CA Telon definition source member name in the CA Endevor Software Change Manager library.

VSQUAL

High level data set name qualifier for the TDF VSAM files into which the definition will be imported.

NDVRENV

CA Endevor Software Change Manager environment

NDVRSYS

CA Endevor Software Change Manager system

NDVRSUB

CA Endevor Software Change Manager subsystem

NDVRTYP

CA Endevor Software Change Manager type

NDVRSTG

CA Endevor Software Change Manager stage for retrieval

NDVRCID

CA Endevor Software Change Manager CCID (change control identification)

NDVRCMT

CA Endevor Software Change Manager Comment

DL/I DBD Importing

DL/I databases referenced in a CA Telon definition within the TDF must be defined in TDF Option 2, Data Administration. CA Telon provides a Database Definition (DBD) import facility to assist you in creating the DL/I data definitions. You can use DBD importing to bring new DL/I DBDs into the TDF and to modify DBDs that have previously been defined to the TDF. All DBD importing is from DBD source code.

However, importing a modified DBD can have significant impact on application definitions that already exist and can reference the spe database. Consequently, although DBD importing is similar to CA Telon definition importing, there are differences that restrict the automation of certain types of DBD changes.

DBD importing automates modification importing and reports on changes that impact existing application definitions.

Example

A major change to the hierarchy of the segments defined in the database typically invalidates the SSA usage in existing database accesses.

This type of change requires manual updates to the Option 2 data definitions and/or the Option 4 and/or 5 application definitions that reference them. However, it is more likely that adding a search field to a database will have no impact on existing application definitions and can be processed automatically by CA Telon DBD importing.

DBD importing is handled in much the same way as definition importing as follows:

- A DBD being imported can be compared with its TDF definition if one exists in TDF Data Administration
- The TDF Data Administration data is *optionally* updated to reflect the contents of the importing DBD source
- The result of all DBD comparison and import processing is reported using the same format as definition import reporting
- The severity codes assigned to the messages are the same as for definition importing (see the later subject "Severity Codes" for a complete description of severity codes)

Processing

DBD import processing has four run types available. The TDF updating associated with each run type is quite different for DBD importing. The DBD import run types are:

C

COMPARE DBD with Data Administration data

O

OVERLAY Data Administration data

M

MERGE DBD with Data Administration data

I

IGNORE Data Administration data

Run Type C (Compare)

Run type **C** compares the importing DBD against its corresponding TDF definition, if one exists. Differences between the two are reported, and major differences, such as a significant change in segment hierarchical structure, are assigned a severity code of **16**. When the DBD being imported does not exist on the TDF, CA Telon terminates processing and produces a message indicating there is no data for comparison. In all cases, the DBD itself is never imported, since this is a COMPARE run only. Consequently there is *no TDF updating* associated with this run type.

Run Type O (Overlay)

Run type **O** replaces the DL/I database definition in its entirety while leaving the CA Telon-specific data intact where possible. The intent is to upgrade the DBD without losing the DLIDSCs (SSAs), which have been defined for use by existing application definitions. Like run type **C**, type **O** runs compare the DBD source being imported with the TDF Data Administration data if the DBD already exists on the TDF. If differences are encountered, they are reported.

If the MAXSEVR parameter value is greater than or equal to the highest encountered severity, the DL/I-specific data is overlaid on the TDF by the imported DBD source. If any message resulting from the overlay has a severity code greater than the MAXSEVR parameter, the data is not overlaid.

In OVERLAY mode, Data Administration data used by CA Telon applications, but not inherent to the DBD, remains intact unless the resultant DBD cannot support the data. For example, if a search field is deleted, any DLIDSCs (SSAs) that reference it are now invalid.

Note: DBD importing with a run type of **O** does not cause the deletion of DL/I database segments. However, it can cause segments to be added to the DL/I database. Consequently, DL/I databases that have segments added as a result of input processing should be reviewed for structural integrity.

Although major segment hierarchical differences result in a severity **16** message, CA Telon allows you to import a DBD with these differences as long as you import the DBD with a run type of **O** and a severity of **20**. This may require manual updating of TDF Option 2, 4, and/or 5 specifications. This type of change typically dictates that existing application definitions be modified for the new database structure.

CA strongly recommends that CA Telon database usage cross-reference reports be produced to assist in determining which application definitions may be impacted by the database modification. (Please refer to the *Implementation* manual for additional information.)

Refer to DBD importing with run type **I** for details on a more automated approach to importing DBDs with major hierarchical changes. The following table summarizes the TDF updating associated with **O** (OVERLAY) type DBD importing runs.

Allowed	Encountered	Data Admin Updated?
0	0	Y
	> 0	N
2	< = 2	Y
	> 2	N
4	< = 4	Y
	> 4	N
8	< = 8	Y
	> 8	N
12	< = 12	Y
	> 12	N
16	< 16	Y
	= 16	N

Run Type M (Merge)

Run type **M** performs the same compare and report processing as run types **C** and **O**. Like run type **O**, TDF Data Administration DBD definitions are replaced by the importing source's information when the maximum allowable severity is not exceeded.

The only difference between run types **M** and **O** is the severity associated with the discrepancies encountered. Run type **M** indicates that the intent of the import is to add information to the existing DL/I DBD. This should have minimal, if any, effect on existing application definitions.

Therefore, if discrepancies are encountered that are likely to impact existing definitions, they are reported with a higher severity so that the maximum severity is exceeded and no DBD import processing occurs. For example, if a DL/I search field type changes, an **O** run type would assign a severity code of **12** to this discrepancy, while a run type of **M** would assign a severity code of **16** to the same condition.

Note: DBD importing with a run type of **M** does not delete DL/I database segments. However, it may add segments to the DL/I database. Consequently, DL/I databases that have segments added as a result of input processing should be reviewed for structural integrity. Additionally, run type **M** does not replace a search field name. It will be added as a new SRCHFLD at the end of the DBD.

The following table summarizes the TDF updating associated with **M** (MERGE) type DBD import runs.

Allowed	Encountered	Data Admin Updated?
0	0	Y
	> 0	N
2	< = 2	Y
	> 2	N
4	< = 4	Y
	> 4	N
8	< = 8	Y
	> 8	N
12	< = 12	Y
	> 12	N
16	< 16	Y
	= 16	N

Run Type I (Ignore)

Run type **I** performs no comparison during import when the DBD already exists on the TDF. That is, an **I** run type import aborts when the DBD is found to have been previously defined to TDF Data Administration.

If the DBD is not already on the TDF, the DBD imports. Thus, run type **I** should be used for DBD imports if you want to guarantee that current TDF defined DBDs are never changed. Run type **I** is only capable of *adding* DBDs to the TDF. The following table outlines the TDF update processing associated with **I** (IGNORE) type DBD import runs.

Max Severity Allowed	Max Severity Encountered	If DBD Not Already In Data Admin
0	N/A	Y
2	N/A	Y
4	N/A	Y
8	N/A	Y
12	N/A	Y
14	N/A	Y

Summary

The following table summarizes the update processing to be performed by import for DBD importing.

Runtype	No Severe Error Encountered	Severe Error Encountered
C-COMPARE	Data Administration data compared and differences reported	Data Administration data compared and differences reported
M-MERGE*	■ Data Administration data compared and differences ■ Data Administration data replaced by importing source data definitions ■ Data Administration data solely related to CA Telon application definitions are not modified using M ■ Report of import results	■ Data Administration data compared and differences reported ■ Data Administration not updated

Runtype	No Severe Error Encountered	Severe Error Encountered
I-IGORE	■ Data Administration data not compared ■ Data Administration data created if it did not exist ■ Data Administration data not created or updated if item already exists ■ Report of import results	Not applicable
O-OVERLAY*	■ For DBD IMPORT ONLY ■ Data Administration data compared to DBD replaced from DBD source if there are differences ■ Data Administration data solely related to CA Telon are not modified using O ■ Report of import results	■ Data Administration data compared and differences reported ■ Data Administration not updated

Note: MERGE and OVERLAY processing are identical, but severities assigned during MERGE import are generally higher.

Importing from a PDS

Run an OS/390 or z/OS Batch job to import a DBD in CA Telon. The model JCL provided in the INSTALL data set is JUMDBD, which executes procedure TLNUMDBD. The CA Telon source for this module must be in a PDS.

The most frequently used input parameters to JUMDBD, as defined in TLNUMDBD, are as follows:

RUNTYPE

Import run type:

C

Compare only.

M

Import and merge TDF.

I

(default) Import, unless the DBD already exists on the TDF. Do not compare or change the existing TDF DBDs.

O

Import and merge.

MAXSEVR

Maximum allowable message severity for import to complete. The default is **0**.

SRCMEM

DBD source member name.

SRCLIB

DBD source PDS.

VSQUAL

High level data set name qualifier for the TDF VSAM files into which the DBD will be imported.

DL/I PSB Definition Importing

DL/I Program Specification Block (PSB) definitions can be used when creating or updating CA Telon application program definitions, provided they have been imported into TDF Option 2, Data Administration. CA Telon provides an import facility that accepts DL/I PSB source as input and creates the matching TDF Data Administration data.

Processing

PSB importing is unlike DBD and CA Telon definition importing in that the importing PSB source is *not* compared against any existing TDF Data Administration PSB data. With this in mind, PSB import accepts only two RUNTYPE parameter values:

I

IGNORE

O

OVERLAY

Additionally, the MAXSEVR parameter is *not* used, as *no comparison* processing implies there are no severity codes to report.

Run Type I (Ignore)

Run type **I** provides for import processing that IGNORES the Data Administration data contained in the TDF. This implies that there is no compare or update processing of the TDF Data Administration data. No comparison means no severity codes, so the CA Telon definition always imports. This import type can be used whenever TDF Data Administration changes are not a concern. The following table summarizes the TDF updating associated with IGNORE type imports.

Allowed	Encountered	Imported?	Updated?
0	N/A	Y	N/A
2	N/A	Y	N/A
4	N/A	Y	N/A
8	N/A	Y	N/A
12	N/A	Y	N/A
16	N/A	Y	N/A
20	N/A	Y	N/A

Run Type O (Overlay)

PSB importing using a run type of **O** processes the same as run type **I**, except that it does not generate an error when a matching PSB already exists in the TDF. In this situation, a run type of **O** deletes the current PSB from the TDF and then imports from the PSB source provided.

Job and Procedure—Importing from a PDS

Run an OS/390 or z/OS Batch job to import PSBs in CA Telon. The model JCL provided in the INSTALL data set is JUMPSB, which executes procedure TLNUMPSB. The CA Telon source for this import must be in a PDS.

The most frequently used input parameters to JUMPSB, as defined in TLNUMPSB, are as follows:

RUNTYPE

Import run type:

I (default)

Import unless the PSB already exists on the TDF.

O

Import and overlay TDF

SRCMEM

PSB source member name.

SRCLIB

PSB source PDS.

VSQUAL

High level data set name qualifier for the TDF VSAM files into which the PSB will be imported.

Sample Report

The following example shows a sample of the CA Telon import report. This report is produced regardless of the run type used, although a run type of **I** produces only import status messages (no Data Administration comparison messages).

The numbers in parentheses, on the left side of the sample, refer to descriptions following the report.

TELON DEFINITION IMPORT — DATA ADMINISTRATION REPORT						
(1) CREATED BY JOB: TLNXIMPT						
(2) IMPORT RUN TYPE: MERGE						
(3) RUN STATUS: DEFINITION IMPORTED, MERGE PROCESSING PERFORMED.						
MAX ALLOWABLE SEVERITY 12						
MAX SEVERITY THIS RUN 12						
(4) IMPORT STATUS:						
TCTEST.SD SUCCESSFULLY IMPORTED.						
TCTEST.PD SUCCESSFULLY IMPORTED.						
TCTEST.PI SUCCESSFULLY IMPORTED.						

(5) DB2 TABLE TELON TRPIEMPL COMPARED TO TDF DIFFERENCES FOLLOW						
(6) TLNROW TRGEMPL COMPARED TO TDF NO DIFFER ENCES						
COLUMN ID COMPARED TO TDF NO DIFFERENCES						
COLUMN NAME COMPARED TO TDF NO DIFFERENCES						
(7) COLUMN DOB COMPARED TO TDF NO DIFFERENCES						
COLUMN SEX NOT FOUND ON TDF..... ADDED						
COLUMN DOE COMPARED TO TDF NO DIFFERENCES						
COLUMN DEPARTMENT COMPARED TO TDF DIFFERENCES FOLLOW						
PARAMETER ACTION VALUES						
(8)						
DCLCOL-KEY TDF VALUE USED SEVERITY 08 TDF 01						
SOURCE						
COLUMN HOURLY_RATE COMPARED TO TDF NO DIFFERENCES						
TLNROW ZZEMPL NOT FOUND ON TDF..... ADDED						

DB2 TABLE TELON TRPIEMPL COMPARED TO TDF NO DIFFERENCES						

DL/I DATA BASE TRG0BDV1 ... COMPARED TO TDF DIFFERENCES FOLLOW						
(9) SEGMENT TRGEMPL COMPARED TO TDF DIFFERENCES FOLLOW						
PARAMETER ACTION VALUES						
.....						
SEGLTH MAXBYTES TDF VALUE USED SEVERITY 08 TDF 00600						
SOURCE 04095						
KEYLTH TDF VALUE USED SEVERITY 12 TDF 00009						
SOURCE 00000						
DL/I SSA *DEFAULT COMPARED TO TDF DIFFERENCES FOLLOW						
PARAMETER ACTION VALUES						
.....						
IMSKEY TDF VALUE USED SEVERITY 12 TDF EMPLIKEY						
SOURCE						
DL/I SSA NAMESSA COMPARED TO TDF NO DIFFERENCES						

(10) IMPORT SUMMARY # COMPARED # MATCHED # MISMATCHED # ADDED						
DL/I DATA BASES 1 : 0 1 0						
DB2 TABLES 2 : 1 1 0						
SEQ DATA SETS 0 : 0 0 0						
VSAM DATA SETS 0 : 0 0 0						

Import Report Fields

The lines on this report are as follows:

1. **CREATED BY JOB** – Identifies the batch job name for this import job.
2. **IMPORT RUN TYPE** – Identifies the mode in which the import is operating. The run types available are COMPARE, MERGE, and IGNORE.
3. **RUN STATUS** – Identifies the overall results of this CA Telon definition import. The information displayed here is dependent upon the run type used.

IGNORE run types do *not* perform any Data Administration comparisons, so the run status information consists of only the definition import results. Run types COMPARE and MERGE report definition import results, Data Administration comparisons and possible update results in the run status portion of the report.
4. **IMPORT STATUS** – Identifies that an attempt was made by import to bring in a CA Telon definition. A message is produced for each component of the CA Telon definition import processed, with the result of import's handling of each.
5. CA Telon prints a status message for each data set, DL/I database, and SQL table compared.

Note: Data sets are either VSAM or sequential, while DB2 tables actually represent views and joins as well.
6. CA Telon prints a status message for each TLNROW in a DB2 definition during import when *any* difference is found in *any* component of the SQL table, view, or join being compared.

Note: The row status does not relate to the status of the columns that it contains. Rather, it relates to the row-specific information defined, such as the *name* of the TLNROW.
7. CA Telon prints a status message for each column of a TLNROW when any difference is found in any component of the DB2 table, view, or join being compared.
8. CA Telon prints a detail discrepancy for each TDF parameter whose value in the importing source does not match the value contained in TDF Option 2, Data Administration.

9. CA Telon prints a status message for each component of a DL/I database when any import source values for the database are different from the TDF Option 2 values for the database. Components include SEGMENTS, XDFLDs, LCHILDs, and SSAs.
10. **IMPORT SUMMARY** – Summarizes the comparison processing performed for each CA Telon definition import report. This summary lists the number of:
 - Databases and data sets compared
 - Databases and data sets found to be identical
 - Databases and data sets with differences
 - Databases and data sets added to the TDF from the importing source

Note: Input source databases (DL/I or SQL) that have *no* data access defined for them (for example, all usage parameters are DUMMY for the database) are *NOT* compared against their TDF counterparts and are therefore not reported.

Severity Codes

The severity codes assigned to the messages are described and generically interpreted as shown in the following table.

Severity Code	Description	Interpretation
0	No differences, or database/data set does not exist on the database and can be added.	Generally accepted.
2	Inherited values are different.	Informational messages or inheritance dropped.
4	Source contains SSAs (DLIDSCs) that do not exist in the TDF (the database is not being added).	Warning or a DLIDSC added.
8	Source and TDF Data Administration values differ, however the item in question is not fundamental to the definition of the database or data set.	Severe warning, data definitions were added to.

Severity Code	Description	Interpretation
12	A segment, search field or SSA defined in the source does not exist on the TDF and cannot be added.	Error, Data Administration updating should be closely reviewed.
16	Source and TDF Data Administration values differ or DL/I hierarchical error. The item in question is critical to the definition of the database or data set.	Severe error, inconsistencies exist which should be resolved by the user. Data Administration updating is not recommended.
maximum	The maximum severity encountered by comparison of the importing data to the TDF for that particular entity is reported.	If the severity codes of any compared data is greater than the MAXSEVR parameter, the import is not processed.

Messages

This subject contains a list of the comparisons being performed for CA Telon definition and DBD import processing, the message key associated with each comparison, and the severity of the key. When the importing item is a CA Telon definition and there is a change to an inherited item, CA Telon drops the inheritance and overrides the inherited value (in that definition only) with the CA Telon source value. This results in a severity **2** instead of severity **4**.

CA Telon Definition Import Messages

Code	Description	Severity
0001	DBD not found (added if action indicated)	0
0002	DBD compared	maximum
0003	DB2 table, view or join not found (added if action indicated)	0
0004	DB2 table, view or join compared	maximum
0005	Sequential file not found (added if action indicated)	0
0006	Sequential file compared	maximum
0007	VSAM file not found (added if action indicated)	0

Code	Description	Severity
0008	VSAM file compared	maximum
0009	VSAM file status message	maximum
0010	Sequential file status message	maximum
0011	Segment status message	maximum
0013	Segment not found (added if action indicated)	8
0014	CICS Queue not found (added if action indicated)	0
0015	CICS Queue compared	maximum
0016	CICS Queue status message	maximum
0017	CICS Journal not found (added if action indicated)	0
0018	CICS Journal compared	maximum
0019	CICS Journal status message	maximum
0021	Search field status message	maximum
0023	Search field not found (added if action indicated)	8
0031	DL/I SSA status message	maximum
0051	DB2 TLNROW status message	maximum
0052	DB2 TLNROW not found (added if action indicated)	4
0061	DB2 column status message	maximum
0062	DB2 column added	0
0064	DB2 column not found	12
0072	LCHILD compared	maximum
0073	LCHILD added	8
0081	XDFLD status message	maximum
0083	XDFLD added	4
1000	DL/I segment not found	8
1001	DL/I segment parent not found.	12
1011	DL/I segment PARENT parameters don't match	16
1012	DL/I segment PARENT parameters don't match, NO DATA ACCESS	12

Code	Description	Severity
1021	DL/I segment LENGTH parameters don't match	8
1031	DL/I segment KEYNAME parameters don't match	12
1041	DL/I segment KEYLTH parameters don't match	12
1211	DL/I LCHILD NAME parameters don't match	12
1221	DL/I LCHILD logical DBNAMES don't match	12
1231	DL/I LCHILD POINTER parameters don't match	12
1241	DL/I LCHILD PAIR parameters don't match	12
1251	DL/I LCHILD INDEX parameters don't match	12
1261	DL/I LCHILD RULES parameters don't match	12
1311	DL/I XDFLD NAME parameters don't match	12
1321	DL/I XDFLD SEGMENT parameters don't match	12
1331	DL/I XDFLD CONST parameters don't match	12
1341	DL/I XDFLD FILLER parameters don't match	12
1351	DL/I XDFLD SUBSEQ parameters don't match	12
1361	DL/I XDFLD DDATA parameters don't match	12
1371	DL/I XDFLD NULLVAL parameters don't match	12
1381	DL/I XDFLD EXTRTN parameters don't match	12
1391	DL/I XDFLD SEARCH parameters don't match	12
2000	DL/I search field not found	12
2001	DL/I search field segment not found	12
2011	DL/I search field LENGTH parameters don't match	12
2021	DL/I search field KEYPIC parameters don't match	12
2031	DL/I search field START positions don't match	12
3000	DL/I SSA not found on TDF (added if action indicated)	4
3001	DL/I SSA references a segment not found on TDF	12
3011	DL/I SSA IOAREA parameters don't match	12
3021	DL/I SSA KEY FEEDBACK parameters don't match	12

Code	Description	Severity
3031	DL/I SSA COMMAND CODE parameters don't match	12
3041	DL/I SSA PATH parameters don't match	12
3051	DL/I SSA CONCATENATION parameters don't match	12
3061	DL/I SSA CURRENT parameters don't match	12
3071	DL/I SSA PARENTAGE parameters don't match	12
3081	DL/I SSA OPTION parameters don't match	12
3091	DL/I SSA LOCKED parameters don't match	12
3101	DL/I SSA VARLTH parameters don't match	12
3111	DL/I SSA OFFSET parameters don't match	12
3121	DL/I SSA FIELD NAME parameters don't match	12
3131	DL/I SSA OPCODE parameters don't match	12
3141	DL/I SSA SEGKEY parameters don't match	12
3151	DL/I SSA boolean operators don't match	12
4081	SQL table TYPE parameters don't match	12
5011	SQL TLNROW TYPE parameters don't match	16
6011	SQL column TYPE parameters don't match	12
6021	SQL column DCLCOL ALIAS parameters don't match	12
6031	SQL column DCLCOL KEY parameters don't match	8
6041	DB2 column DCLCOL NOTNULL parameters don't match	8
6051	SQL column DCLCOL TYPE parameters don't match	8
6061	SQL column DCLCOL LENGTH parameters don't match	8
6071	SQL column DCLCOL DECIMALS parameters don't match	8
6081	SQL column DCLCOL ACCESS parameters don't match	8
6091	SQL column DCLCOL join LABEL parameters don't match	8

Code	Description	Severity
6101	TELON SQL join column LABEL1 parameters don't match	8
6111	TELON SQL join column1 parameters don't match	8
6121	TELON SQL join column LABEL2 parameters don't match	8
6131	TELON SQL join column2 parameters don't match	8

DL/I DBD Import Messages

Code	Description	Severity Overlay-Merge
0001	DBD not found (run type changed to IGNORE)	0 - 0
0002	DBD compared	maximum
0011	Segment status message	maximum
0013	Segment added	8 - 8
0021	Search field status message	maximum
0023	Search field added	8 - 8
0031	DL/I SSA status message	12 - 16
0071	LCHILD not found	4 - 8
0072	LCHILD compared	maximum
0073	LCHILD added	8 - 8
0081	XDFLD status message	maximum
0083	XDFLD added	8 - 8
1005	DL/I segment created	8 - 8
1012	DL/I segment PARENT parameters don't match, NO DATA ACCESS	12 - 16
1021	DL/I segment LENGTH parameters don't match	4 - 8
1041	DL/I segment KEYLTH parameters don't match	4 - 8

Code	Description	Severity Overlay-Merge
1131	DL/I segment default PROCOPT parameters don't match	4 - 4
1141	DL/I segment default INDICES parameters don't match	4 - 8
1151	DL/I segment KEY TYPE parameters don't match	4 - 8
1161	DL/I segment KEY FIELD parameters don't match	4 - 8
1171	DL/I segment KEY FIELD NAME parameters don't match	4 - 8
2000	DL/I search field not found	12 - 12
2005	DL/I search field created	8 - 8
2007	DL/I search field TYPE parameters match	4 - 8
2009	DL/I search field deleted	12 - 16
2011	DL/I search field LENGTH don't match	4 - 8
2031	DL/I search field START positions don't match	4 - 4

Code	Description	Severity Overlay-Merge
8001	DBD order error	16 - 16
8002	DBD hierarchical error	16 - 16
8003	XDFLD order error	16 - 16
8004	LCHILD order error	16 - 16
9031	TDF Data Administration enqueue failure	16 - 16

Appendix G: Using Stored Procedures

This section addresses concerns that relate to DB2 stored procedures. The appendix includes information on:

- Designing and developing stored procedures
- Implementing and using CA-Telon generated stored procedures

Generated Programs

The stored procedures functionality provides for the generation of a unique program model for stored procedure programs. In addition, the ability to generate calls to stored procedures from non-stored procedure programs is also provided. Each of these will be discussed.

Stored Procedure Programs

Stored procedures have a unique program model called stored which differs from the screen, batch, report, driver and CICS nonterminal models. By comparison with the others, the stored procedure model is of simpler design. In procedural code for example, there are only three major sections: one for initialization, one for processing, and a third for termination. Given below is a listing of the generated PROCESS-LOOP section in a generated COBOL stored procedure:

```
PROCESS-LOOP
*****
* P R O C E S S L O O P *
*****
PROCESS-LOOP SECTION.
    PERFORM Q-1000-PROGRAM-INIT.
    PERFORM A-1000-STORED-INIT.
    PERFORM C-3000-STORED-PROCESS UNTIL END-OF-TRANSACTION.
    PERFORM D-1000-STORED-TERM.
    PERFORM T-1000-PROGRAM-TERM.
SKIP1
PROCESS-LOOP-RETURN.
EXIT.
```

The A-1000 section moves linkage variables to their mapped working storage counterparts, and the D-1000 section performs the reverse mapping:

A-1000

```
A-1000-STORED-INIT SECTION.
*****
* A - 1 0 0 0 - S T O R E D - I N I T *
*****
* THIS ROUTINE INITIALIZES ANY FIELDS NECESSARY ON *
* ENTRY TO THE STORED PROCEDURE CALL. *
* *
* GENERATED - FIELD INITIALIZATION *
* COPY CODE - STORED/SPINIT *
*****
SKIP1
    MOVE LK-EMPLID TO WS-START-BROWSE-KEY.
    MOVE LK-FLAG TO WS-FLAG.
    MOVE LK-DEBUG TO WS-DEBUG.
* STORED/SPINIT PARAMETER NOT CODED
SKIP1
    A-1000-STORED-INIT-RETURN.
EXIT.
```

D-1000

```
D-1000-STORED-TERM SECTION.
*****
* D - 1 0 0 0 - S T O R E D - T E R M *
*****
* THIS ROUTINE PERFORMS END-OF-PROCESSING FUNCTIONS *
* FOR THE STORED PROCEDURE CALL. *
* *
* COPY CODE - STORED/SPTERM *
*****
SKIP1
* STORED/SPTERM PARAMETER NOT CODED
SKIP1
    MOVE WS-START-BROWSE-KEY TO LK-EMPLID.
    MOVE WS-FLAG TO LK-FLAG.
    MOVE WS-DEBUG TO LK-DEBUG.
    D-1000-STORED-TERM-RETURN.
EXIT.
```

The program may loop through the C-3000 section multiple times, but the default mode is to perform it only once, as follows:

C-3000-STORED-PROCESS

C-3000-STORED-PROCESS SECTION.

```
*****
* C - 3 0 0 0 - S T O R E D - P R O C E S S                *
*****
* THIS SECTION PROVIDES FOR STORED PROCEDURE CALL          *
* CUSTOM CODE. *
*
* COPY CODE - STORED/SPPROC                                *
*****
* STP-LOOP-FLAG = SPACES TERMINATES TRANSACTION            *
*****
```

IF END-OF-TRANSACTION

GO TO C-3000-STORED-PROCESS-RETURN.

SKIP1

MOVE SPACE TO WORKFLD-ALPHA.

MOVE ZERO TO WORKFLD-NUMERIC.

SKIP1

EXEC SQL DECLARE BROWSE_TRGEMPL CURSOR WITH RETURN FOR

```
SELECT EMPL_ID,
       EMPL_NAME,
       EMPL_PHONE,
       EMPL_STREET,
       EMPL_CITY,
       EMPL_STATE,
       EMPL_ZIP,
       EMPL_DEPARTMENT,
       EMPL_HOURLY_RATE,
       EMPL_HOURS
```

FROM TRGEMPL

WHERE ((EMPL_ID >= :DCLTRGEMPL.EMPL-ID))

ORDER BY EMPL_ID

END-EXEC.

IF NOT BROWSE-TRGEMPL-CURSOR-OPEN

EXEC SQL OPEN BROWSE_TRGEMPL

END-EXEC

MOVE CURSOR-OPEN-LIT TO BROWSE-TRGEMPL-CURSOR-IND.

* STORED/SPPROC PARAMETER NOT CODED

MOVE SPACE TO STP-LOOP-FLAG.

SKIP1

C-3000-STORED-PROCESS-RETURN.

EXIT.

As you can see, there are custom code points available in the A-1000, C-3000, and D-1000 sections. In addition, it is possible to define both user exec and autoexec data access requests. BROWSE, TRANSACT and READNEXT data access requests build result sets if the HOLDCUR parameter is set to Y.

In addition to generating the stored procedure, the Generator also builds a CREATE PROCEDURE file allowing the stored procedure to be defined to the DB2 system. Following is an example of a generated CREATE PROCEDURE statement.

```
CREATE PROCEDURE
CREATE PROCEDURE TELON.TRSPCS2K(
    IN EMPLID CHAR(6),
    IN FLAG CHAR(1),
    IN DEBUG CHAR(1))
    DYNAMIC RESULT SETS 1
    EXTERNAL
    LANGUAGE COBOL
    PARAMETER STYLE DB2SQL
    NOT DETERMINISTIC
    FENCED
    MODIFIES SQL DATA
    NO DBINFO
    COLLID TLNSTP
    WLM ENVIRONMENT D710WLM
    COMMIT ON RETURN NO
    ASUTIME NO LIMIT
    STAY RESIDENT NO
    PROGRAM TYPE MAIN
    SECURITY DB2;
```

Note: Both COBOL and PL/I stored procedures can be built with CA Telon.

The HOLDCUR is set to Y for the READNEXT data access request, the generator generates a DECLARE ... CURSOR WITH RETURN. This in turn produces a result set for the calling program. If HOLDCUR is not set to Y, a valid data access request is still generated, but its effects are limited to the stored procedure itself. No result set will be generated for the calling program.

The calling program must specify a FETCH that is consistent with the READNEXT specified in the stored procedure. Given in the following is what is generated in the U-100-NEXT-TRGEMPL section:

```

U-100-NEXT-TRGEMPL.
    EXEC SQL DECLARE NEXT_TRGEMPL CURSOR WITH RETURN FOR
        SELECT EMPL_ID,
            EMPL_NAME,
            EMPL_PHONE,
            EMPL_STREET,
            EMPL_CITY,
            EMPL_STATE,
            EMPL_ZIP,
            EMPL_DEPARTMENT,
            EMPL_HOURLY_RATE,
            EMPL_HOURS
        FROM TRGEMPL
        WHERE (EMPL_ID >= :WS-START-BROWSE-KEY)
        ORDER BY EMPL_ID
    END-EXEC.
IF NOT NEXT-TRGEMPL-CURSOR-OPEN
    PERFORM U-100-NEXT-TRGEMPL-OPEN.

U-100-NEXT-TRGEMPL-OPEN.
    EXEC SQL OPEN NEXT_TRGEMPL
    END-EXEC.
MOVE CURSOR-OPEN-LIT TO NEXT-TRGEMPL-CURSOR-IND.
IF SQLCODE NOT = 0 AND SQLCODE NOT = -502
    AND SQLCODE NOT = +100
    MOVE SQLCODE TO TRGEMPL-STATUS
    PERFORM U-100-SET-DA-STATUS-DB2
        MOVE 0 TO ABT-ERROR-ABEND-CODE
        MOVE 'OPEN' TO ABT-DA-FUNCTION
        MOVE 'DB2 ' TO ABT-ERROR-ACTIVITY
        MOVE 'TRGEMPL ' TO ABT-DA-ACCESS-NAME
        CALL 'ADMAAB0'
    PERFORM Z-980-ABNORMAL-TERM.

```

Note: The stored procedure must open the cursor, but not close it. Although a DECLARE ... CURSOR is generated, the corresponding FETCH logic is not; this code needs to be generated in the calling program.

Programs Which Call Stored Procedures

In addition to generating the stored procedures themselves, CA Telon can generate calls to stored procedures in other program models. The user defines the stored procedures to be called by the program, and the information about these procedures is generated as the following code fragment illustrates:

S-100-STP-CALLS SECTION.

```

*****
* S - 1 0 0 - S T P - C A L L S *
*****
* THIS ROUTINE INITIALIZES ANY FIELDS NECESSARY ON *
* ENTRY TO THE STORED PROCEDURE CALL. *
* * *
* COPY CODE - STORED/PRESP *
* COPY CODE - STORED/POSTSP *
* GENERATED - STORED PROCEDURE CALL AND MAPPINGS *
*****
*
  S-100-CALL-TRSPCS2K.
*

*      STORED INITIAL CC POINT/PRESP COPY CODE

*COPY: C:\DEV\PWS41S\COBSRC\TLNWORK\MANRI02\102019692.7\PRESP.COB LVL1
      MOVE WS-START-BROWSE-KEY TO TRSPCS2K-ID.
*END: C:\DEV\PWS41S\COBSRC\TLNWORK\MANRI02\102019692.7\PRESP.COB LVL1
      MOVE 0 TO CURSOR-TRSPCS2K-1-FLAG.
      EXEC SQL
          CALL TRSPCS2K(:TRSPCS2K-ID, :TRSPCS2K-FLAG)
      END-EXEC.
      IF SQLCODE NOT = +466
          MOVE 3508 TO ABT-ERROR-ABEND-CODE
          MOVE 'CALLSTOR' TO ABT-DA-FUNCTION
          MOVE 'DB2 ' TO ABT-ERROR-ACTIVITY
          MOVE 'TRSPCS2K' TO ABT-DA-ACCESS-NAME
          CALL 'ADMAAB0'
          PERFORM Z-980-ABNORMAL-TERM.
*
      EXEC SQL ASSOCIATE LOCATORS (:LOC-TRSPCS2K-1)
          WITH PROCEDURE TRSPCS2K
      END-EXEC.

* STORED POST STP CALL CC POINT/POSTSP COPY CODE

*COPY: C:\DEV\PWS41S\COBSRC\TLNWORK\MANRI02\102019692.7\POSTSP.COB LVL1
      MOVE TRSPCS2K-FLAG TO WS-FLAG.
*END: C:\DEV\PWS41S\COBSRC\TLNWORK\MANRI02\102019692.7\POSTSP.COB LVL1
*
  S-100-STP-CALLS-RETURN.
      EXIT.

```

Note: The PRESP and POSTSP custom code points would typically be used to map program variables to/from the calling parameters for the stored procedure call.

If any result sets are produced by the call to the stored procedure, the S-200-STP-CURSORS section is generated. This section contains one S-200-CURSOR-stored-procedure-N paragraph for each result set. Each of the paragraphs allocates the cursor associated with the corresponding result set. If no result sets are expected, the S-200 section is not generated, and the ASSOCIATE LOCATORS command is not issued in the S-100 section.

On return from a call to a stored procedure the normal SQLCODE when one or more result sets is produced is +466. The default logic treats any other SQLCODE as a significant error. It is possible, however, to specify IGNORE codes to deal with other possible outcomes, or even to do away with the SQLCODE check altogether (IGNORE=ALL). Following is an example of a typical S-200 section:

S-200-STP-CURSORS SECTION.

```
*****
* S - 1 0 0 - S T P - C A L L S *
*****
* THIS ROUTINE CONTAINS CURSOR PROCESSING FOR ALL *
* RESULT SETS FOR ALL STORED PROCEDURE CALLS. *
* *
* COPY CODE - STORED/RESLTCC *
* GENERATED - CURSOR PROCESSING *
*****
*
S-200-CURSOR-TRSPCS2K-1.
*
      IF CURSOR-TRSPCS2K-1-FLAG = 0
      EXEC SQL ALLOCATE CURSOR-TRSPCS2K-1 CURSOR
              FOR RESULT SET :LOC-TRSPCS2K-1
      END-EXEC
      MOVE 1 TO CURSOR-TRSPCS2K-1-FLAG.
*
      MOVE CURSOR-OPEN-LIT TO CURSOR-TRSPCS2K-1-IND.
*
*
      STORED RESULT SET CC POINT/RESLTCC COPY CODE
      SKIP1
*COPY: C:\DEV\PWS41S\COBSRC\TLNWORK\MANRI02\102019692.7\RESLTCC.COB LVL1
      IF SQLCODE NOT = 0
      MOVE 'E' TO WS-FLAG.
*END: C:\DEV\PWS41S\COBSRC\TLNWORK\MANRI02\102019692.7\RESLTCC.COB LVL1
*
S-200-STP-CURSORS-RETURN.
EXIT.
```

Three data access requests can be specified in a calling program to FETCH the result sets produced by data access requests in the stored procedure:

Request in calling program...	FETCHes result set from request in stored procedure
SPBROWSE	BROWSE
SPRDNEXT	READNEXT
SPTRNACT	TRANSACT

If multiple result sets are retrieved in the calling program, each must have a unique result number (RESLTNR) associated with it. A custom code exit point RESULTCC is provided, located in the ALLOCATE ... CURSOR command in the S-200 section.

The U-100-READNEXT section in such a case will be generated as follows:

U-100-NEXT-TRGEMPL.

```

EXEC SQL  FETCH  CURSOR-TRSPCS2K-1
          INTO   :DCLTRGEMPL.EMPL-ID,
                  :DCLTRGEMPL.EMPL-NAME,
                  :DCLTRGEMPL.EMPL-PHONE,
                  :DCLTRGEMPL.EMPL-STREET,
                  :DCLTRGEMPL.EMPL-CITY,
                  :DCLTRGEMPL.EMPL-STATE,
                  :DCLTRGEMPL.EMPL-ZIP,
                  :DCLTRGEMPL.EMPL-DEPARTMENT,
                  :DCLTRGEMPL.EMPL-HOURLY-RATE
          :TRGEMPL-EMPL-HOURLY-RATE-NN,
          :DCLTRGEMPL.EMPL-HOURS
          :TRGEMPL-EMPL-HOURS-NN
          END-EXEC.
MOVE SQLCODE TO TRGEMPL-STATUS.
MOVE 'Y' TO DB2-COMMIT-READ-IND.
PERFORM U-100-SET-DA-STATUS-DB2.
IF TRGEMPL-STATUS = +100
    PERFORM U-100-NEXT-TRGEMPL-CLOSE
ELSE
IF TRGEMPL-STATUS NOT = 0
    AND TRGEMPL-STATUS NOT = +100
    MOVE 3507          TO ABT-ERROR-ABEND-CODE
    MOVE 'READNEXT'    TO ABT-DA-FUNCTION
    MOVE 'DB2 '        TO ABT-ERROR-ACTIVITY
    MOVE 'TRGEMPL '    TO ABT-DA-ACCESS-NAME
    CALL 'ADMAAB0'
    PERFORM Z-980-ABNORMAL-TERM.

```

Note: There is no DECLARE or OPEN of the cursor in the stored procedure. The S-100 and S-200 sections access the stored procedure, then associate a cursor for the result set, so that the calling program may proceed directly to the FETCH logic, closing the cursor when a +100 SQLCODE is found.

Calling "Foreign" Stored Procedures

You can define stored procedures to CA Telon that have been built outside of CA Telon. Use the DB2 Stored Procedure catalog extract to retrieve needed information from the DB2 catalog. The catalog extract creates a transport file which can be transported into the CA Telon Design Facility database. Following is the format of the transport control card used to transport a stored procedure into the TDF:

```
//*.....0.....5.....0.....5.....0.....5.....0.....5.....0  
STORED schema storedprocedure hhiiii
```

The schema name is placed in columns 10-17; the stored procedure name, in columns 18-35. The stored procedure name can have up to 18 characters with no systematic naming convention, although there is a limit of 8 characters for COBOL and PL/1 calling programs. The TDF requires a Header/ID format to define a program entity, so you must assign a meaningful header and ID for each stored procedure in columns 36-41.

When adding a "foreign" stored procedure to a calling program in the TDF you must specify the Header/ID, and not the stored procedure name.

Glossary

Auto Exec

Generated Data Access where both the code to handle the data access and the code to determine where it is executed are generated automatically by CA Telon.

Automatic I/O

I/O routines that are automatically set up by CA Telon and occur every time the program executes. See *Auto Exec* above.

Batch Definition

A description in the TDF of all the specifications that make up a batch program. It is created with the Batch Definition Facility.

BATCH

A CA Telon statement that specifies the characteristics of a CA Telon Batch Program Definition.

BOTPAGE

A CA Telon Batch Panel Image group defining the data to be printed on the bottom of a physical page.

BROWSE

A type of Auto Exec that generates multiple direct and sequential reads for execution of online List Screen processing.

CA Telon Code

Source code statements generated by the TDF. They are an intermediate step in the generation of a PL/I or COBOL program. This is referred to both as CA Telon code and CA Telon source code.

CA Telon Design Facility (TDF)

The CA Telon online, interactive tool used to create CA Telon source code, the first step in generating a program with CA Telon.

CA Telon Generator

The CA Telon batch utility that produces COBOL or PL/I source code from CA Telon and custom source code.

CA Telon Source Code

Source code statements generated by the TDF. They are an intermediate step in the generation of a PL/I or COBOL program. This is referred to both as CA Telon code and CA Telon source code.

CICSBMS

A CA Telon statement that requests the generation of a BMS map.

CICSPGM

A CA Telon statement that requests the generation of a CICS program.

CINCRE

A SEGLOOP statement specifying the incremental number of columns between data groups on a screen.

Circle Flow Diagram

A diagram of screen-to-screen flow in an application. It is called circle flow since it often is circular in design. This diagram is also referred to as a Screen Flow diagram.

CONSIG

A SCREEN statement parameter identifying Custom Code to be executed during input consistency edits.

Consistency Edits

CA Telon or user-generated logic that assures that one or more fields are consistent with other data in the system.

CONTROL

A CA Telon Batch Panel Image group defining data to be printed when specific data changes during printing.

CPYCALL

A DB2 Auto Exec and User Exec option that allows the customization of generated SQL statements and their inclusion in the paragraph or procedure generated for the User Exec.

CPYINIT

A DB2 Auto Exec and User Exec option that allows the inclusion of Custom Code at the beginning of the paragraph or procedure generated for the User Exec.

CPYKEY

A DB2 Auto Exec and User Exec option that allows the inclusion of the generated WHERE logic in the SQL statement generated for the User Exec.

CPYTERM

A DB2 Auto Exec and User Exec option that allows the inclusion of Custom Code at the end of the paragraph or procedure generated for the User Exec.

CREATE

A CA Telon statement that requests the generation of a section or procedure that creates a DL/I segment or VSAM record. You must execute the resulting code from your own Custom Code. This I/O is not performed automatically by the generated CA Telon program.

CURSCUS

A SCREEN statement parameter identifying Custom Code used to determine cursor positioning when the cursor is not positioned at the same location each time the screen is displayed.

Custom Code

Your own COBOL or PL/I source code that is inserted unchanged by CA Telon into the generated program.

Custom Data Access

Data access statements that are not generated by CA Telon but are included in the Custom Code of a program. You should not use custom data access unless absolutely necessary.

Data Selection Criterion

A CA Telon grouping of information necessary to format an SSA, such as the IMS search field name, key length and op codes. In addition, it includes the value of the KEY field for an Auto Exec or User Exec statement.

DATABAS

A CA Telon statement that defines the DL/I data base used by the program. It generates the program PSB.

DATA SET

A CA Telon statement that defines the VSAM file used by a CICS program.

DB2

The name of IBM's new relational data base. It is also the name used by CA Telon for a DB2 table when defined in Generated Data Access.

DCLCOL

Defines the columns in the TLNROW statement preceding the DCLCOL statement.

DELETE

A CA Telon statement that requests the generation of a section or procedure that deletes a DL/I segment or VSAM record. You must execute the resulting code from your own Custom Code.

Design-time Modeling

The process of creating an interactive model to review a design while the system is still in the design stage.

DETAIL

A CA Telon Batch Panel Image group defining the data to be printed in the main body of the report.

DLIPSB

A CA Telon statement that requests the generation of a DL/I PSB for the program.

Driver Definition

A description in the CA Telon Design Facility (TDF) of all specifications that make up an IMS Driver program. It is created with the TDF.

DRIVER

A CA Telon statement that specifies the characteristics of an IMS/DC driver program.

ENDTRAN

A MATCH statement parameter identifying Custom Code performed in C-1000-GET-END-TRAN. This identification takes place after reading all records in a batch of Match Transaction records (i.e., all records, rows, or segments with matching Match keys).

Evolutionary Development

An extension of design-time modeling in which a series of ever more complex applications is created. The model of one cycle is the source for the next.

FIELD

A CA Telon statement that defines the characteristics of the fields in a CA Telon-generated program.

FMTCUST

A GROUP statement parameter identifying Custom Code performed at the end of the B-5000-FORMAT-groupname section/procedure to make custom changes to the group format in a batch program.

FMTEXTIT

A CA Telon FIELD statement parameter specifying the MFS field exit routine to be invoked for the field on input.

Generated Data Access

Data access that is generated by CA Telon from high level non-procedural statements.

GETTRAN

A BATCH or NONTERM statement parameter identifying Custom Code performed for non-automatic input processing in the C-1000-GET-TRAN or C-N100-GET-TRAN section/procedure.

GROUP

A CA Telon statement that identifies the beginning of a number of FIELD statements that refer to a single logical event in the batch program (for example: the TOPPAGE group, the DETAIL group).

GROUPEND

A BATCH statement identifying the end of a batch group. A batch group is a number of batch statements beginning with the GROUP statement.

Help Facility

A TDF facility with online information about using the TDF. CA Telon makes it easy for you to create a similar facility for your application.

hhPCBs

The user-maintained copy member defining the PCB structure of an IMS program running under TSO.

hhPROC

A user-maintained copy member containing the argument list of PCBs or PROCs used by the program.

hhUPDTA

A user-maintained copy member defining the common update work area used with automatic DL/I segment updating.

hhWKAREA

A user-maintained copy member defining the common work area used by a CA Telon-generated application.

HOLD Data Base

For IMS/DC and TSO target environments, the data base used for HOLD processing. A HOLD data base is defined by specifying the Auto Exec type as HOLD for that data base.

HOLD Facility

A CA Telon facility that allows you to *hold* one screen temporarily while using a different screen. This means you can make a HOLD request, go to another part of the application, and later return to the same screen (or position in the screen) you were on when you issued the HOLD request. CA Telon makes it easy for you to create a similar facility for your application.

ICUST1

A SEGLOOP statement parameter used as an exit for Custom Code prior to each iteration of input edits (in the loop).

ICUST2

A SEGLOOP statement parameter identifying Custom Code executed during list processing at the end of each iteration of input. ICUST2 replaces the ICUSTOM parameter.

IMSDRV

A CA Telon statement that requests the generation of an IMS COBOL or PL/I online driver program.

IMSMFS

A CA Telon statement that requests the generation of IMS MFS DIF/DOF, MID, and MOD control blocks for the screen.

IMSPGM

A CA Telon statement that requests the generation of an IMS COBOL or PL/I screen program or report subroutine.

IMSPSB

A CA Telon statement that requests the generation of an IMSPSB.

INCRE

A SEGLOOP statement parameter specifying the incremental number of lines between data groups on a screen.

ININIT

A SCREEN statement parameter identifying Custom Code performed during D-100-INPUT-EDIT after any automatic file access.

INIT1

A BATCH statement parameter identifying Custom Code performed before the file OPEN statements (if any) in Q-1000-PROGRAM-INIT in a CA Telon batch program. INIT1 can also appear before the printer verification (if any) in Q-N100-PROGRAM- INIT in a CICS Nonterminal program.

INIT2

A BATCH statement parameter identifying Custom Code performed at the end of Q-1000-PROGRAM- INIT (after the file OPEN statements, if any) in a CA Telon batch program. INIT2 can also appear at the end of Q-N100-PROGRAM-INIT (after the printer verification, if any) in a CICS Nonterminal program.

INMAST

A MATCH statement parameter identifying Custom Code performed in C-1000-GET-MAST. This identification takes place after each successful read of a Match Master Autoexec.

INPUT

A CA Telon FIELD statement parameter defining a field that transfers user-entered data from the screen to the program.

INREAD

A type of Auto Exec used to generate a read during input processing for an online program.

INTERM

A SCREEN statement parameter identifying Custom Code performed during H-100-INPUT-TERM after any automatic I/O.

INTRAN

A MATCH statement parameter identifying Custom Code performed in C-1000-GET-TRAN. This identification takes place after each successful read of a Match Transaction Autoexec.

ISEGIDX

A SEGLOOP statement input parameter identifying the index name for the array referenced in each SEGLOOP iteration.

JOINCOL

For DB2, the CA Telon statement parameter used to define the join criteria for a CA Telon table join. JOINCOL values define the column names (and the corresponding table names) used to generate SQL statements for an Auto Exec or User Exec defined for the table join.

LINECNT

A SEGLOOP statement parameter specifying the name of an output field on the screen that is to display the line number.

Literal Break Character

A character used on the Panel Image input screen to divide closely placed Literal fields into separate fields.

LITERAL

A CA Telon FIELD statement parameter defining a field that simply informs the operator of screen or field content.

MAPOUT

A FIELD statement parameter used to designate OUTPUT or OUTIN fields that are mapped for output only when certain conditions are met.

MATCH

A CA Telon statement that specifies the characteristics of a Batch MATCH. Also, a MATCH statement parameter identifying Custom Code in A-1000-MAST-EQ-TRAN. This Custom Code is performed from A-1000-MATCH whenever the keys from a Transaction and a Master file are equal.

MATCHEND

A CA Telon statement indicating the end of MATCH processing statements.

MATCHKEY

A CA Telon statement that specifies the characteristics of a Batch MATCH Key.

MATCHX

A type of Auto Exec used to generate Sequential Reads for Batch Match transactions.

MERGE

A CA Telon statement that specifies the characteristics of a Batch MERGE (either Mainline or User-defined).

MERGE#

A type of AUTO EXEC used to generate Sequential Reads for Batch Merge transactions.

MERGEEND

A CA Telon statement indicating the end of MERGE processing statements.

MERGEGRP

A CA Telon statement that specifies the characteristics of a Batch MERGE Key Group.

MERGEKEY

A CA Telon statement that specifies the characteristics of a Batch MERGE Key.

MGREATR

A MATCH statement parameter identifying Custom Code in A-1000-MAST-GREATER. This Custom Code is performed from A-1000-MATCH whenever the keys from the Master record are greater than the keys of the Transaction record.

NEXTPGM

A FIELD and a SCREEN statement parameter used to identify the next program to be executed.

NONTERM

A CA Telon statement that specifies characteristics of a CA Telon CICS Nonterminal definition.

Nonterminal Definition

A description in the TDF of all specifications that make up a CICS Nonterminal program.

OCUST1

A SEGLOOP statement parameter identifying Custom Code executed during list processing after the first data base or data set call and before the fields are mapped for output.

OCUST2

A SEGLOOP statement parameter identifying Custom Code executed during list processing after each subsequent data base or data set call.

OCUST3

A SEGLOOP statement parameter identifying Custom Code executed during list processing just before the output line is displayed.

OINIT1

A SCREEN statement parameter identifying Custom Code performed before data files are automatically processed for output.

OINIT2

A SCREEN statement parameter identifying Custom Code performed after data files are automatically processed for output.

OIREAD

A type of Auto Exec used to generate a read during both output processing and input processing for an online program.

OSEGIDX

A SEGLOOP statement parameter identifying the index name for an array from which output values are to be mapped in each SEGLOOP iteration.

OUTIN

A CA Telon FIELD statement parameter defining a field that both transfers data from a program to the terminal and returns the data from the terminal to the program.

Output/Input Architecture

Structure of CA Telon-created programs where data is first moved to the screen and then returned to the program.

OUTPUT

A CA Telon FIELD statement parameter defining a field that transfers data from the program to the terminal or a report.

OUTREAD

A type of Auto Exec used to generate a read during output processing for an online program.

OUTTERM

A SCREEN statement parameter identifying Custom Code performed at the end of the B-100-OUTPUT-EDITS section/procedure.

Panel Definition

A description in the TDF of all specifications relative to the fields that are mapped to and from the terminal. It is created with the Panel Definition Facility or Panel Specification.

Panel Image

The TDF layout of a screen, a report, or batch events. It is created from the Panel Definition Facility or Panel Specification.

PFKEYS

A SCREEN statement parameter identifying Custom Code performed as the first code processed, when input processing begins. This normally includes your definition of the programs response to PF key usage.

PGMCUST

A SCREEN, BATCH, or NONTERM statement parameter identifying Custom Code to be placed before, after, or in place of CA Telon-generated COBOL sections or PL/I procedures.

PLIXOPT

A CA Telon statement used to override installation-defined PL/I defaults.

PRCTRAN

A BATCH or NONTERM statement parameter identifying Custom Code performed at the end of A-1000-PROCESS-TRAN or A-N100-PROCESS-TRAN to override default transaction processing.

Presentation Stores

An unstructured collection of sample data values used with the Prototyping Facility.

Program Definition

A description in the TDF of all specifications that make up a screen, report, driver, or batch program.

PRTDEST

A NONTERM statement parameter that identifies the printer ID where you will route a Nonterminal report.

READ

A CA Telon statement that requests the generation of a section or procedure that reads a DL/I segment or VSAM record. You must execute the resulting code from your own Custom Code.

READNEXT

A type of User Exec used to generate sequential read paragraphs or procedures.

REPEAT

A SEGLOOP statement parameter specifying the number of times the INCRE is repeated for a group of data.

Report Definition

A description in the TDF of all the specifications that make up a report subroutine. It is created with the Program Definition Facility or Panel Specification.

Report Events

One of six different activities a CA Telon Batch program can perform on a report output. They are TOPPAGE, TOPDTL, DETAIL, CONTROL, BOTPAGE, and SUMMARY.

REPORT

A CA Telon statement that specifies the characteristics of a Report Definition.

Row

For DB2, one occurrence of the columns in a table. Conceptually, a row is similar to a record or segment occurrence.

SCONSIS

A FIELD statement parameter identifying Custom Code to be executed during consistency edits of Select fields.

SCPRINT

A CA Telon statement that requests a printed screen image for testing and debugging purposes.

Screen Definition

A description in the TDF of all the specifications that make up a screen program. It is created with the Program Definition Facility.

Screen Flow Diagram

A diagram of screen-to-screen flow in an application.

SCREEN

A CA Telon statement that specifies the characteristics of a Screen Definition.

SECTION

A SCREEN, BATCH, and NONTERM statement parameter identifying custom code inserted as a separate PL/I procedure or COBOL section at the end of the program. You must execute this code from your own Custom Code.

SEGEDIT

A CA Telon statement used to implement an automatic consistency check between screen entry fields and data base or data set data.

SEGEND

A CA Telon statement indicating the end of SEGLOOP list processing statements.

SEGLOOP

A CA Telon statement defining the beginning of a series of statements specifying the characteristics of a CA Telon listing process.

SEGMENT

A CA Telon statement that defines the DL/I segment or VSAM file access information required by the program.

SELECT

A FIELD statement parameter defining a field that acts as a special input field. The input field forces the application user to designate the next processing to be done by the program.

SORT

A CA Telon statement that specifies the characteristics of a Batch SORT (either Mainline or User-defined).

SORTEND

A CA Telon statement indicating the end of SORT processing statements.

SORTKEY

A CA Telon statement that specifies the characteristics of a Batch SORT Key (for either a Mainline or User-defined sort).

SRC

A CA Telon statement used to add a single line of Custom Code before or after a SEGEDIT or XFEDIT consistency edit. You can use several SRC statements in an SQL row.

STRUCTURE

A BATCH statement parameter identifying the batch program's structure. Valid values for this parameter are SORT (i.e., the program is a Mainline Sort), MATCH (i.e., the program is a Match), and MERGE (i.e., the program is a Merge). If the parameter is not coded, the program structure is Standard.

SUMMARY

A CA Telon Batch Panel Image group defining data to be printed at the end of the report.

Table

For DB2, a collection of rows and columns. Conceptually, a table is similar to a file or data base.

TDF

The CA Telon Design Facility: a CA Telon utility allowing online, interactive parameter entries to create CA Telon source code.

TELON

A CA Telon statement defining the version of CA Telon you want to use when overriding the installation default version.

For other CA Telon-related terminology, see terms starting with CA Telon.

TERM

A BATCH or NONTERM statement parameter identifying Custom Code performed just before closing files in T-1000-PROGRAM-TERM or T- N100-PROGRAM-TERM.

TGREATR

A MATCH statement parameter identifying Custom Code in A-1000-TRAN-GREATER. This Custom Code is performed from A-1000-MATCH whenever the keys from the Transaction record are greater than the keys of the Master record.

TLNDSC

For DL/I, the CA Telon statement used to define Data Selection Criterion. A Data Selection Criterion contains the information necessary to format an SSA, such as the IMS search field name, key length, and op codes. In addition, it includes the value of the KEY field for an Auto Exec or User Exec statement.

TLNJOIN

For DB2, the CA Telon statement used to define information for the CA Telon join of two or more DB2 tables.

TLNROW

For DB2, the CA Telon statement used to define information for a DB2 table row.

TOPDTL

A CA Telon Batch Panel Image group that defines data to be printed at the top of a Detail group.

TOPPAGE

A CA Telon Batch Panel Image group that defines data to be printed at the top of the page.

TPPCB

A CA Telon statement that requests the generation of a teleprocessing PCB in the PSB of an IMS CA Telon program.

TRANSACTION

A type of Auto Exec for Batch and CICS Nonterminal processing used to generate sequential reads for the input transactions.

Transaction

A record area or areas that drive one iteration of a batch process creating one or more report output lines. It is a record(s), segment(s), message queue (IMS/DC), JCL parm, or interface with another program.

TSOPGM

A CA Telon statement that requests the generation of an online TSO COBOL or PL/I program.

UPDATE

A CA Telon statement that requests the generation of a section or procedure that reads a DL/I segment or VSAM record. You must execute the resulting code from your own Custom Code.

User Exec

Generated Data Access in which the code to handle the data access is generated, but not the code to determine where it is executed. A User Exec creates a paragraph or procedure that is performed or called from Custom Code.

User I/O

I/O operations that are under the control of Custom Code. This allows you to selectively execute the code depending on program conditions. See User Exec above.

WKAREA

A SCREEN statement and BATCH statement parameter identifying Custom Code to be inserted into the program's work area definition. This code is in addition to the standard hhWKAREA copy member.

WORKSPA

A type of Auto Exec for IMS/DC online processing used to handle accesses to a work file to simulate conversational processing.

XFEDIT

A CA Telon statement used to implement an automatic consistency edit to compare two or more fields in the program and on the screen.

XFERWKA

A SCREEN statement parameter identifying Custom Code inserted into the program as the program's Transfer Work Area.

Index

A

- A-1000-PROCESS-TRAN program section • 176
 - indicator • 176
- is reset • 176
- A4MSG parameter • 446
 - IMSPGM Statement • 446
- A4EPGM parameter • 446
 - IMSPGM Statement • 446
- ABCALL parameter • 478
 - TPPCB Statement • 478
- ACCESS parameter • 433
 - DCLCOL Statement • 433
- Active Presentation Store • 318, 322, 323, 325
 - adding fields to • 325
 - clearing • 322
 - definition of • 323
 - deleting fields from • 325
 - displaying variables • 318
 - loading • 322
 - saving • 318
- ALARM parameter • 460
 - SCREEN Statement • 460
- ALIAS parameter • 433
 - DCLCOL Statement • 433
- ALIGN parameter • 456
 - PLIXOPT Statement • 456
- APPLID parameter • 437, 453, 460
 - DRIVER Statement • 437
 - NONTERM statement • 453
 - SCREEN Statement • 460
- Array variables • 318
 - subscripting • 318
- attributes • 250, 251, 253
 - extended • 250, 251
 - overriding • 250
- ATTRINT parameter • 439
 - FIELD Statement • 439
- ATTRPRO parameter • 439
 - FIELD Statement • 439
- Auto Exec • 61, 69, 71, 76, 77
 - creating a program definition • 69
 - creating a program definition, batch • 76
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61

- Auto Exec requests • 410, 412, 414, 415, 416, 417
 - journals • 412
 - Match Master(MATCHM) • 410
 - Match Transaction(MATCHT) • 410
 - Merge(MERGE) • 410
- automatic line numbering • 256
 - in SEGLOOP processing • 256
- auxiliary storage • 423
 - queues • 423

B

- Batch • 53, 54, 56, 57, 61, 190
 - adding processing logic • 54
 - output • 54
 - processing logic • 56
 - program control • 190
 - report design • 53
 - specifying a Panel Image • 53
 - transaction design • 53
- batch characteristics • 76
 - creating a program definition, batch • 76
- Batch Definition • 63, 77, 98, 359, 361, 366, 368, 370, 375, 377, 385, 390, 393
 - Creating the • 359
 - definition of • 63
 - Program Definition structure, batch • 77
 - TDF Main menu • 98
- Batch Mainline • 174
 - B-2000-END-REPORT • 174
 - C-1000 GET TRAN • 174
 - MAIN-PROCESS-LOOP • 174
 - Q-1000-PROGRAM-INIT • 174
 - T-1000-PROGRAM-TERM • 174
- BATCH statement, parameters • 174, 176, 190
 - FMTCUST • 176
 - GETTRAN • 190
 - PARMS • 174
 - PRCTRAN • 174, 190
 - TERM • 174
- Batch, data base/data set definition • 54
 - with reports • 54
- BOOLEAN parameter • 437
- BOTPAGE • 73, 74
 - Batch Panel Definition • 74
 - definition of • 73

- browsing • 256
 - OPCODE • 256
 - SCHFLDC • 256
 - SCHFLDI • 256
 - SCHFLDL • 256

C

- CALC parameter • 439
 - FIELD Statement • 439
- CAPS parameter • 460
 - SCREEN Statement • 460
- CICS • 401, 403
 - initializing applications • 401
- CICS Client Program Hierarchical Charts • 194
 - main processing • 194
- CICS Server Program Hierarchical Charts • 204
 - main processing • 204
- CICS Server Program Structure • 207
 - options • 207
- CINCRE parameter • 466
 - SEGLOOP Statement • 466
- CMDCODE parameter • 464
 - SEGEDIT Statement • 464
- CNTGRP parameter • 439
 - FIELD Statement • 439
- COBOL • 63
 - ANSI • 63
- COLSGLP parameter • 466
 - SEGLOOP Statement • 466
- columns • 354
 - on line • 354
- Compare run type • 657, 666
 - DL/I DBD importing • 666
 - TELON definition importing • 657
- COND parameter • 480
 - exceptions • 480
 - XFEDIT Statement • 480
- Condition codes • 424
 - IGNORE for queues and journals • 424
- CONDPTR parameter • 442
 - GETDIAG Statement • 442
- CONSIS parameter • 460, 482
 - SCREEN Statement • 460, 482
- Consistency Edits • 61, 63, 68, 71, 74, 76, 77, 129, 141, 145, 146, 148, 151, 236, 244, 247, 248, 281, 482, 483, 539, 540, 541, 544, 546, 547, 548, 555, 556, 557, 559, 563, 564, 567, 568, 571, 572, 576, 583, 587, 588, 593, 594, 595, 596, 597, 598, 601, 602, 603, 605, 607, 608, 614, 618, 619, 623
 - controlling processing flow • 281
 - creating a Panel Definition • 68, 129
 - Custom Code • 482
 - custom coded • 236
 - defining the • 141
 - error processing • 236
 - field-to-data base example • 244
 - field-to-field example • 244
 - hierarchical approach • 236
 - logic • 483
 - non-procedural edit structure • 236
 - Panel Definition, batch • 74
 - position in program • 236
 - Program definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
 - program structure • 61
 - relationship with Panel Definition • 63
 - SEGEDIT • 236
 - TELON-generated edit structure • 236
 - XFEDIT • 236
- Consistency edits, custom coded • 236
 - CONSIS • 236
 - SCONSIS • 236
- consistency edits, error processing • 244
- CONTROL • 73, 74
 - Batch Panel Definition • 74
 - definition of • 73
- CONTROL-INDICATOR • 158, 160, 161, 194, 195, 197, 204, 206, 207
 - CONTINUE-PROCESS-LIT • 161, 197, 206
 - DO-TRANSFER-LIT • 161, 197
 - DO-WRITE-LIT • 158, 194, 204, 206
 - INIT-FORM-LIT • 206
 - PROCESS-FORM-LIT • 206
 - PROCESS-INPUT-LIT • 161, 197
 - PROCESS-OUTPUT-LIT • 161, 197
- controlling flow • 313
 - in prototyping • 313
- controlling processing flow • 253
 - screen to screen • 253
- controlling screen flow • 274
 - with PF keys • 274

- CONVERS parameter • 446
 - IMSPGM Statement • 446
- CONVERT parameter • 230, 439
 - example • 230
 - FIELD statement • 230, 439
 - format • 230
- COPY code • 274
 - PF keys • 274
- COPY members • 281, 393, 400
 - defining for data base segment • 400
 - hhPCBs • 393
 - hhPROC • 393
 - hhUPDTA • 393
 - hhWKAREA • 393
 - SCONSIS • 281
 - XFERWKA • 393
- COPY parameter • 457, 469, 476
 - RECORD statement • 457
 - SORT statement • 469
 - TABLE Statement • 476
- COPYLBL parameter • 457, 469, 476
 - RECORD statement • 457
 - SORT statement • 469
 - TABLE Statement • 476
- COPYLV1 parameter • 457, 469, 476, 477
 - RECORD statement • 457
 - SORT statement • 469
 - TABLE Statement • 476
- CREATE command • 326
 - in prototyping • 326
- Creating a Panel Definition • 74
 - batch • 74
- creating canned scenarios • 333, 335
 - example • 333
- CTLLTH parameter • 443
 - GROUP Statement • 443
- CTLPIC parameter • 443
 - GROUP Statement • 443
- CTLVAR parameter • 443
 - GROUP Statement • 443
- CURSCUS parameter • 162, 208, 460
 - SCREEN statement • 162, 208, 460
- CURSOR parameter • 162, 208, 460, 464, 480
 - SCREEN statement • 162, 208, 460
 - SEGEDIT Statement • 464
 - XFEDIT Statement • 480
- cursor positioning • 248
 - automatic • 248
 - CURSOR-ATTR • 248

- CURSOR-BLANK-ATTR • 248
- ERROR-ATTR • 248
 - fields used • 248
 - special I/O • 248
 - user-defined • 248
- cursor positioning , user-defined • 248
 - initial definition • 248
- cursor positioning, automatic • 248
 - in edit processing • 248
- cursor positioning, user-defined • 248, 250
 - using FLDEDIT • 248
- custom code • 61, 63, 66, 69, 71, 76, 77, 221, 224, 248, 482, 483
 - basics of using • 221
 - Consistency Edits • 482
 - creating a program definition • 69
 - creating a program definition, batch • 76
 - cursor positioning • 248
 - definition of • 63
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
 - X-100-CONISIS-EDITS • 482
- Custom code, parameters • 224, 230
 - CONISIS • 224
 - CURSCUS • 224
 - FLDEDIT • 224
 - ICUSTOM • 224
 - ININIT2 • 224
 - OCUST1 • 224
 - OCUST2 • 224
 - OCUST3 • 224
 - OINIT1 • 224
 - OINIT2 • 224
 - OUTTERM • 224
 - PFKEYS • 224
 - PGMCUST • 224
 - SCONSIS • 224
 - WKAREA • 224

D

- data access • 61, 63, 69, 71, 76, 77
 - creating a program definition • 69
 - creating a program definition, batch • 76
 - definition of • 63
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
- Data Administration • 98

- TDF Main Menu • 98
- Data Base/Data Set Definition • 56
 - Batch • 56
 - without reports • 56
- DATA SET statement, parameters • 174
 - OPEN • 174
- data transfer • 403
 - storage area • 403
- DATABAS statement, parameters • 393
 - DBDNAME • 393
 - PCBNAME • 393
- DATANAME parameter • 452, 453
 - MERGEKEY statement • 452
- DB2 • 98
 - read-only access • 98
- DBNAME parameter • 439
 - FIELD Statement • 439
- DCLCOPY parameter • 476
 - TABLE Statement • 476
- DCLHOST parameter • 433
 - DCLCOL Statement • 433
- DCLLBL parameter • 476
 - TABLE Statement • 476
- DCLRDEF parameter • 476
 - TABLE Statement • 476
- DEC parameter • 433
 - DCLCOL Statement • 433
- Declaring Global Temporary Tables • 640
 - with Create/Update SQL Table (D141) • 640
 - with Select New Row Name (S137) • 640
- defaults • 105
 - User Profile Maintenance menu • 105
- defining PF Keys • 105
 - User Profile Maintenance menu • 105
- DESC parameter • 437, 452, 453, 460, 464, 465, 466, 469, 480
 - DRIVER Statement • 437
 - MERGEGRP statement • 452
 - NONTERM statement • 453
 - SCREEN Statement • 460
 - SEGEDIT statement • 464
 - SORT statement • 469
 - XFEDIT statement • 480
- DESCR parameter • 476
 - TABLE Statement • 476
- DETAIL • 73, 74
 - Batch Panel Definition • 74
 - definition of • 73
- detail data • 61, 68, 71, 76, 77

- Program definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
- determining where to pass control • 281
 - using SELECT field • 281
- DIAGS parameter • 442, 443
 - GETDIAG Statement • 442
- Driver Definition • 63, 69, 410
 - creating a program definition • 69
 - definition of • 63

E

- EACOLOR parameter • 439
 - FIELD Statement • 439
- EAERR parameter • 460
 - SCREEN Statement • 460
- EAHIGH parameter • 439
 - FIELD Statement • 439
- EAIN parameter • 460
 - SCREEN Statement • 460
- EALIT parameter • 107, 109, 111, 460
 - SCREEN Statement • 460
 - Update Program Definition Defaults • 107
- EAVAILID parameter • 439
 - FIELD Statement • 439
- EDIT command • 327
 - in prototyping • 327
- edit condition • 480, 482
 - expanded version, COND parameter • 480
- Edit Panel Image, screen • 53
 - Batch • 53
- environment • 61, 63, 69, 71, 76, 77
 - creating a program definition • 69
 - creating a program definition, batch • 76
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
- ENVIRONMENT parameter • 456
 - PLIXOPT Statement • 456
- EOFKEY parameter • 298, 308
 - SCREEN statement • 298, 308
- ERRCHAR parameter • 464, 480
 - SEGEDIT Statement • 464
 - XFEDIT Statement • 480
- ERRMSG parameter • 464, 480
 - SEGEDIT Statement • 464
 - XFEDIT Statement • 480
- Error messages • 158, 194, 204, 332, 333, 335
 - creating • 332
 - defaults • 332

- in prototyping • 335
- sending • 158, 194, 204
- ERROR parameter • 464
 - SEGEDIT Statement • 464
- error processing • 281, 286, 288, 289, 293, 332
 - example • 281
 - simulating • 332
- Executing CA Telon • 630, 631
 - developed applications • 630, 631
- bind DB2 • 631
- pre-compile DB2 • 630
- specifying DBRM members • 630
- exiting the TDF • 318
 - in prototyping • 318
- EXPRESS parameter • 478
 - TPPCB Statement • 478

F

- FETCH Details Screen (S244) • 636, 639
 - FETCH for nn rows • 639
 - using alternate cursor • 636
- FETCH for nn rows • 639
 - FETCH Details Screen (S244) • 639
- field characteristics • 61, 68, 71, 74, 76, 77, 129
 - creating a Panel Definition • 68, 129
 - creating a program definition, batch • 74
 - Program definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
- field edits • 63, 230
 - generated logic • 230
 - relationship with Panel Definition • 63
- field names • 333
 - special • 333
- FIELD statement, parameters • 167, 230, 253, 256, 281, 407
 - CONVERT • 230
 - FLDTYPE • 230
 - FORMAT • 230
 - HELPMMSG • 167
 - IEXTEND • 230
 - INDBIO • 281
 - INEDIT • 281
 - MAPOUT • 230
 - NEXTPGM • 253, 281
 - OEXTEND • 230
 - PIC • 230
 - RANGE • 230
 - REQ • 230
 - SCONSIS • 281

- SELKEY • 256, 281, 407
- VALUES • 230
- field types • 73
 - batch • 73
- FILEIN parameter • 469
 - SORT statement • 469
- FILEOUT parameter • 469
 - Sort statement • 469
- FLDTYPE parameter • 230, 313, 439
 - ALPHA value • 230
 - FIELD Statement • 439
 - format • 230
 - in prototyping • 313
 - NONE value • 230
- flow charts, TDF screens • 105, 313
 - Prototyping Facility • 313
 - User Profile Maintenance • 105
- flow control • 313
 - in prototyping • 313
- FMTCNTL parameter • 439
 - FIELD Statement • 439
- FMTCUST parameter • 176, 177, 178, 179, 443
 - BATCH statement • 176
 - GROUP statement • 176, 443
- FMTEXIT parameter • 439
 - FIELD Statement • 439
- FORGRP parameter • 443
 - GROUP Statement • 443
- form initialization processing • 207, 208
 - CICS Server • 207
- FORM parameter • 452, 470
 - MERGEGRP statement • 452
 - SORTKEY statement • 470
- FORMAT parameter • 230, 439
 - example • 230
 - FIELD statement • 230, 439
 - syntax • 230
- FROM parameter • 476, 477, 478
 - TABLE Statement • 476
 - TLNJOIN Statement • 477
- FRSTPGM parameter • 437
 - DRIVER Statement • 437
- FUNC parameter • 434, 464
 - DELETE Statement • 434
 - SEGEDIT Statement • 464
- FUNCTION parameter • 100, 105
 - 0.1 TDF Main menu • 100

G

- GDCLSC parameter • 442
 - GETDIAG Statement • 442
- GDCUST parameter • 442
 - GETDIAG Statement • 442
- GDOPNC parameter • 442
 - GETDIAG Statement • 442
- GENKEYL parameter • 434, 464
 - DELETE Statement • 434
 - SEGEDIT Statement • 464
- GENPCBS parameter • 446, 479
 - IMSPGM Statement • 446
 - TSOPGM Statement • 479
- GETTRAN parameter • 190
 - BATCH statement • 190
- GROUP statement, parameters • 176
 - FMTCUST • 176
 - SKIPBEF • 176
- GRPISRT parameter • 424

H

- HEADER parameter • 437, 453
 - DRIVER Statement • 437
 - NONTERM statement • 453
- HELP command • 318
 - in prototyping • 318
- HELP parameter • 162, 198
 - SCREEN statement • 162, 198
- HELP processing • 280, 281
 - example • 280
 - return • 280
 - return example • 280
- HELP text • 295
 - for IMS/DC • 295
- HELPMMSG parameter • 162, 167, 198, 199, 200, 201, 439
 - FIELD statement • 162, 167, 198, 439
- HIGHLIGHT parameter • 464, 480
 - SEGEDIT Statement • 464
 - XFEDIT Statement • 480
- HOLD parameter • 162, 198, 437
 - DRIVER Statement • 437
 - SCREEN statement • 162, 198
- HOLD processing • 318, 320, 321
 - in prototyping • 318
- HOLDLTH parameter • 444
 - HOLD Statement • 444
- host variables • 281, 306, 307
 - CONSIG-INPUT-ERRORS • 281

- CONTROL-INDICATOR • 281
- ERROR-ATTR • 281
- ERROR-MESSAGE-MULTHIT • 281
- ERROR-MESSAGE-NOHIT • 281
- LINK-WORK-AREA • 306
- SPA-XFER-WORK-AREA • 306
- TPO-ERRMSG1 • 281

I

- ICTLNM parameter • 256, 466
 - SEGLOOP statement • 256, 466
- ICUST1 parameter • 466
 - SEGLOOP Statement • 466
- ICUST2 parameter • 466
 - SEGLOOP Statement • 466
- ICUSTOM parameter • 167, 212, 215, 216, 256
 - SEGLOOP statement • 167, 212, 256
- ID parameter • 437
 - DRIVER Statement • 437
- IEXTEND parameter • 439
 - FIELD Statement • 439
- ignore run type • 669
 - DL/I DBD importing • 669
- importing from a PDS • 661, 670, 671, 673, 676, 677
 - DL/I DBD • 670
 - TELON definition • 661
- importing from PANVALET • 663
 - TELON definition • 663
- IMS/DC • 393
 - initializing applications • 393
- INCRE parameter • 256, 466
 - SEGLOOP statement • 256, 466
- INDBIO parameter • 167, 281, 439
 - FIELD statement • 167, 281, 439
- INEDIT parameter • 281, 335, 439
 - FIELD statement • 281, 439
 - in prototyping • 335
- ININIT1 parameter • 167, 212
 - SCREEN statement • 167, 212
- ININIT2 parameter • 167, 212
 - SCREEN statement • 167, 212
- INIT parameter • 437, 439
 - DRIVER Statement • 437
 - FIELD Statement • 439
- input • 115
 - field type • 115
- input mapping • 336
 - in prototyping • 336

- input processing • 159, 166, 167, 171, 173, 174, 201, 202, 204, 212, 274
 - C-600-FORM-PROCESS program section • 202
 - CONTAINS-NO-SELECT-FIELDS • 167
 - CONTAINS-SELECT-FIELDS • 167
 - D-100-INPUT-INIT • 167, 212
 - normal sequence • 274
 - P-100-PFKEYS program section • 166, 201
 - Set DO-TRANSFER • 171, 204
- input processing, in SEGLOOP processing • 256, 274
 - array example • 256
- INTENSITY • 316, 317, 318
 - Prototyping Facility • 316
- IOAREA parameter • 449, 464
 - JOURNAL statement • 449
 - SEGEDIT Statement • 464
- ISEGIDX parameter • 256, 466
 - SEGLOOP statement • 256, 466
- ITEM parameter • 424
 - Data Access I/O statement • 424
- ITEMLBL parameter • 424
 - Data Access I/O statement • 424

J

- JFILEID parameter • 423
 - CJOURNAL statement • 423
- JOINLBL parameter • 433, 434
 - DCLCOL Statement • 433
- Journals • 412, 424
 - Auto Exec requests • 412
 - condition codes, IGNORE • 424
 - data access I/O statement • 412
 - generation of • 412
 - statement • 412
 - User Exec requests • 412
- JREF1 parameter • 449
 - JOINCOL Statement • 449
- JREF2 parameter • 449
 - JOINCOL Statement • 449
- JTYPEID parameter • 423
 - CJOURNAL statement • 423

K

- KEY parameter • 433, 459, 464
 - DCLCOL Statement • 433
 - ROW Statement • 459
 - SEGEDIT Statement • 464

- KEYLEN parameter • 405
 - SEGMENT statement • 405
- KEYPIC • 256
 - paging on a DL/I segment • 256

L

- LABEL parameter • 439, 443, 464, 469, 476, 477, 478
 - FIELD Statement • 439
 - GROUP Statement • 443
 - SEGEDIT Statement • 464
 - SORT statement • 469
 - TABLE Statement • 476
 - TLNJOIN Statement • 477
 - TLNROW Statement • 478
- LANG parameter • 437
 - DRIVER Statement • 437
- LANGLVL parameter • 453, 454, 477
 - NONTERM statement • 453
 - TELON Statement • 477
- LAST-LINENO host variable • 405
 - in loop processing • 405
- line numbers • 256
 - in SEGLOOP processing • 256
 - in SELECT field • 256
- LINECNT parameter • 256, 466
 - SEGLOOP statement • 256, 466
- LINEOPT parameter • 307, 446
 - CICSPGM statement • 307
 - IMSMFS Statement • 446
 - IMSPGM statement • 307, 446
- LINKOPT parameter • 446
 - IMSPGM Statement • 446
- LINKPGM parameter • 446
 - IMSPGM Statement • 446
- List screen • 329
 - prototyping • 329
 - types • 329
- literal field type • 73
 - batch • 73
- LNKCOPY parameter • 446, 479
 - IMSPGM Statement • 446
 - TSOPGM Statement • 479
- LOCFLAG parameter • 442
 - GETDIAG Statement • 442
- LRECL parameter • 423
 - CJOURNAL statement • 423
- LRECL1 parameter • 469
 - SORT statement • 469

- LRECL2 parameter • 469
 - SORT statement • 469
- LTERM parameter • 304, 306, 478
 - TPPCB statement • 304, 478
- LTH parameter • 433, 439, 452, 470
 - DCLCOL Statement • 433
 - FIELD Statement • 439
 - MERGEGRP statement • 452
 - SORTKEY statement • 470
- LTHLBL parameter • 449
 - JOURNAL statement • 449
- LTHOPT parameter • 449, 457
 - JOURNAL statement • 449
 - RECORD statement • 457

M

- MAINAUX parameter • 423
 - CQUEUE statement • 423
- Mainline • 160, 161, 174, 196, 197, 205
 - batch • 174
 - online hierarchical chart • 160, 196, 205
- Mainline Sort • 410, 419, 469
 - BATCH statement, STRUCTRE parameter • 419
 - data base storage • 410
 - generation of • 410, 469
 - statement • 410, 469
- MAPOUT parameter • 230, 439
 - FIELD statement • 230, 439
- mapping • 256
 - to an array • 256
- mapping from an array • 329, 330, 331, 332
 - in prototyping • 329
- Mapping, in prototyping • 336, 338, 339
 - command entry • 336
- mapping, input • 336
 - in prototyping • 336
- Match Sort • 410, 412, 419, 424, 450
 - Auto Exec requests • 410
 - BATCH statement, STRUCTRE parameter • 419
 - data access I/O statement • 412, 424
 - generation of • 410
 - statement • 450
 - User Exec requests • 412, 424
- MAXSEVR parameter • 653, 656
 - parameter • 653
- Merge run type • 658, 668
 - DL/I DBD importing • 668

- TELON definition importing • 658
- Merge Sort • 410, 412, 419, 424, 451, 452
 - Auto Exec request • 412
 - BATCH statement, STRUCTRE parameter • 419
 - data access I/O statements • 412, 424
 - generation of • 410
 - statement • 410, 451
 - User Exec requests • 412
 - USEREXEC requests • 412
- MFSMOD parameter • 446
 - IMSMFS Statement • 446
 - IMSPGM Statement • 446
- MGREATR parameter • 450, 451
 - MATCH statement • 450
- MINOR parameter • 443
 - GROUP Statement • 443
- MSGBUF parameter • 446
 - IMSPGM Statement • 446
- MSGCALL parameter • 478
 - TPPCB Statement • 478
- MSGPGM parameter • 446
 - IMSPGM Statement • 446
- MSGTBL parameter • 446
 - IMSPGM Statement • 446
- MSGTRAN parameter • 446
 - IMSPGM Statement • 446

N

- NAME parameter • 423, 476, 477, 478
 - CQUEUE statement • 423
 - TABLE Statement • 476
 - TLNJOIN Statement • 477
 - TPPCB Statement • 478
- NBR parameter • 442
 - GETDIAG Statement • 442
- NEXTPGM parameter • 321, 322, 439
 - FIELD Statement • 439
 - in prototyping • 321, 322
- NEXT-PROGRAM-NAME-ID key word • 321
 - in prototyping • 321
- Nonterminal Definitions • 63, 98, 411, 412, 453
 - characteristics • 63
 - Custom Code, editing • 453
 - driver of • 411
 - generated programming language • 453
 - generation of • 411
 - printer specification for • 453
 - report, length and width • 453

- statement for • 411, 453
- TDF Main Menu • 98
- TDF name • 63
- Nonterminal Program • 411
 - parameters of • 411
 - statement for • 411
- NOSUSP parameter • 449
 - JOURNAL statement • 449
- NOTNULL parameter • 433
 - DCLCOL Statement • 433
- O**
- OCUST1 parameter • 224, 466
 - SEGLOOP statement • 224, 466
- OCUST2 parameter • 466
 - SEGLOOP Statement • 466
- OCUST3 parameter • 466
 - SEGLOOP Statement • 466
- OEXTEND parameter • 439
 - FIELD Statement • 439
- OF parameter • 439
 - FIELD Statement • 439
- OINIT1 parameter • 162, 208
 - SCREEN statement • 162, 208
- OINIT2 parameter • 162, 208
 - SCREEN statement • 162, 208
- Online Program Hierarchical Charts • 158
 - main processing • 158
- Online Program Structure • 162, 198
 - options • 162, 198
- OPCODE parameter • 256, 434, 464
 - DELETE Statement • 434
 - SEGEDIT Statement • 464
 - SEGMENT statement • 256
- OPEN parameter • 174, 432
 - DATA SET statement • 174
- OSEGIDX parameter • 466
 - SEGLOOP Statement • 466
- OUTATTR parameter • 439
 - FIELD Statement • 439
- OUTIFIL parameter • 298, 308
 - SCREEN statement • 298, 308
- outin • 115
 - field type • 115
- output • 115
 - field type • 115
- output field • 68
 - 01 to 79 character error message field • 68
- output field type • 73

- batch • 73
- output processing • 162, 198
 - online • 162, 198
- OUTTERM parameter • 464
 - SCREEN Statement • 464
- overlay run type • 666, 672
 - DL/I DBD importing • 666
 - DL/I PSB importing • 672

P

- PAGE parameter • 256, 405, 466
 - SEGLOOP statement • 256, 405, 466
- PAGESAV parameter • 256, 405, 466
 - SEGLOOP statement • 256, 405, 466
- paging • 256
 - with PF keys • 256
- Panel definition • 61, 63, 68, 71, 74, 76, 77, 129
 - Batch • 74
 - creating a, batch • 74
 - Creating the • 129
 - definition of • 63
 - Program definition structure • 61, 71
 - Program Definition structure, batch • 77
 - structure of • 68
 - structure of, batch • 76
- Panel Definition, updating • 318
 - in prototyping • 318
- panel field characteristics • 61
 - field data detail, program structure • 61
- Panel Image • 61, 63, 66, 68, 69, 71, 76, 77, 113, 115
 - creating a • 66, 113
 - definition of • 63
 - Program definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
 - program structure • 61
- Panel Image, updating • 318
 - in prototyping • 318
- Panel Specification • 98
 - TDF Main menu • 98
- PANVALET • 663, 665
 - importing from • 663
- PARMS parameter • 174
 - BATCH statement • 174
- PCBNAME parameter • 464
 - SEGEDIT Statement • 464
- PF key program section • 274
 - other uses • 274
- PF Keys • 105, 274, 278, 280

-
- COPY code name • 274
 - how to define • 105
 - in HELP processing • 280
 - in HOLD processing • 278
 - processing conventions • 274
 - RESUME logic • 278
 - to abort processing • 274
 - to control screen flow • 274
 - PF keys, in HOLD processing • 278
 - example • 278
 - PFKEY-RETURN-INDICATOR host variable • 274
 - setting • 274
 - PFKEYS parameter • 464
 - SCREEN Statement • 464
 - PFXLBL parameter • 449
 - JOURNAL statement • 449
 - PFXLTH parameter • 449
 - JOURNAL statement • 449
 - PGMCUST parameter • 437, 453, 458, 464
 - DRIVER Statement • 437
 - NONTERM statement • 453
 - REPORT Statement • 458
 - SCREEN Statement • 464
 - PGMNAME parameter • 444, 446
 - IMSDRV Statement • 444
 - IMSPGM Statement • 446
 - PIC parameter • 230, 236, 439
 - FIELD statement • 230, 439
 - syntax • 230
 - PKYLTH parameter • 466
 - SEGLOOP Statement • 466
 - PKYUNIQ parameter • 466
 - SEGLOOP Statement • 466
 - PL/I • 63
 - OS • 63
 - POS parameter • 439
 - FIELD Statement • 439
 - PRCTRAN parameter • 174, 190, 194
 - BATCH statement • 174, 190
 - PREFIX parameter • 469
 - SORT statement • 469
 - Presentation Store • 326, 327, 328
 - creating • 327
 - loading • 326
 - retrieving • 326
 - saving • 326
 - Presentation Stores • 313, 328, 329, 333, 340
 - definition • 313
 - listing • 328
 - merging • 333
 - naming conventions • 340
 - PRINT parameter • 302, 443, 478, 479
 - GROUP Statement • 443
 - TPPCB statement • 302, 478
 - PROCESS-FORM processing • 212
 - P-100-PFKEYS program section • 212
 - processing • 280, 281
 - HELP request • 280
 - using SELECT logic • 281
 - PROCIN parameter • 469
 - SORT statement • 469
 - PROCOUT parameter • 469
 - Sort statement • 469
 - program control • 190
 - batch • 190
 - Program Definition • 98, 359
 - Creating the • 359
 - TDF Main menu • 98
 - Program Structure Charts • 158
 - prototyping • 48, 49, 50, 313, 315
 - field editing • 313
 - flow control • 313
 - purpose of • 48
 - Prototyping Facility • 98, 318, 322, 323, 332, 333, 335, 336, 339, 340, 341, 352, 354
 - Commands • 318
 - control hints • 340
 - displaying special fields • 333
 - error messages • 335
 - Examples • 340
 - input mapping • 336
 - simulating an error • 332
 - simulating special fields • 339
 - TDF Main menu • 98
 - updating fields • 322
 - using TDF commands • 318
 - Prototyping Facility menu, screen, parameters • 323, 325
 - PROTOTYPE LEVEL • 323
 - PRTDEST parameter • 453
 - NONTERM statement • 453
 - PSBNAME parameter • 437, 448
 - DLIPSB Statement • 437
 - IMSPSB Statement • 448
 - Public Access • 633
 - Plans • 633
-

Q

- QUAL parameter • 476, 477
 - TABLE Statement • 476
 - TLNJOIN Statement • 477
- QUALIFY parameter • 464
 - SEGEDIT Statement • 464
- QUELBL parameter • 457
 - RECORD statement • 457
- QUETYPE parameter • 423
 - CQUEUE statement • 423
- Queues • 412
 - generation of • 412
 - statement • 412

R

- RANGE parameter • 230, 439, 442
 - example • 230
 - FIELD statement • 230, 439
 - format • 230
- RECLTH parameter • 434
 - DELETE Statement • 434
- REFRESH parameter • 298, 309, 464
 - SCREEN statement • 298, 309, 464
- REMARKS parameter • 437, 453, 458, 464
 - DRIVER Statement • 437
 - NONTERM statement • 453
 - REPORT Statement • 458
 - SCREEN Statement • 464
- REORDER parameter • 456
 - PLIXOPT Statement • 456
- report characteristics • 61, 69, 71, 77
 - creating a program definition • 69
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
- Report Definition • 61, 63, 69, 71
 - creating a program definition • 69
 - definition of • 63
 - Program Definition structure • 61, 71
 - program structure • 61
- REPSEQ parameter • 443
 - GROUP Statement • 443
- REQ parameter • 230
 - FIELD statement • 230
 - format • 230
 - REQ=C • 230
- REQID parameter • 449
 - JOURNAL statement • 449
- REQIDLBL parameter • 449

- JOURNAL statement • 449
- reserved fields • 256, 274, 281
 - CONSIS-INPUT-ERRORS • 281
 - ERROR-ATTR • 281
 - ERROR-MESSAGE-MULTHIT • 281
 - ERROR-MESSAGE-NOHIT • 281
 - for input processing • 256
 - NEXT-PROGRAM-NAME-ID • 274
 - PFKEY-INDICATOR • 274
 - SEGLOOP-COUNT • 256
 - TPO-ERRMSG1 • 281
- RPTDEST parameter • 453
 - NONTERM statement • 453

S

- SAVEKEY parameter • 405, 466
 - SEGLOOP statement • 405, 466
- SCHFLDC parameter • 256, 405, 466
 - SEGLOOP statement • 256, 405, 466
- SCHFLDI parameter • 256, 405, 466
 - SEGLOOP statement • 256, 405, 466
- SCHFLDL parameter • 256, 405, 466
 - SEGLOOP statement • 256, 405, 466
- SCONSIS code • 281
 - with tailored edit logic • 281
- screen characteristics • 61, 69, 71, 77
 - creating a program definition • 69
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
- screen concept • 36, 37, 38, 39, 40
 - circle diagram • 38
 - components • 36
- Screen Definition • 61, 63, 69, 71, 77, 359
 - creating a program definition • 69
 - Creating the • 359
 - definition of • 63
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
- screen flow charts, TDF screens • 105
 - User Profile Maintenance • 105
- Screen Flow diagram • 40, 44, 47, 48
 - conventions • 40
- screen flow diagrams • 313
 - Prototyping Facility • 313
- Screen organization • 83, 98, 105, 313
 - nonterminal definitions • 83
 - Prototyping Facility • 313

- User Profile Maintenance • 105
- SCREEN statement, parameters • 162, 163, 165, 166, 167, 208, 209, 210, 212, 253, 274, 280, 281, 298, 301, 308, 309
 - CURSCUS • 162, 208
 - CURSOR • 162, 208
 - EOFKEY • 298, 308
 - ININIT1 • 167, 212
 - ININIT2 • 167, 212
 - NEXTPGM • 253, 281
 - OINIT1 • 162, 208
 - OINIT2 • 162, 208
 - OUTIFIL • 298, 308
 - PFKEYS • 274, 280
 - REFRESH • 298, 309
 - SECTION • 281
- SECTION parameter • 281, 437, 453, 458, 464
 - DRIVER Statement • 437
 - NONTERM statement • 453
 - REPORT Statement • 458
 - SCREEN statement • 281, 464
- SEGEDIT • 61, 68, 71, 76, 77
 - Program Definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
- SEGEDIT statement, parameters • 236
 - CURSOR • 236
 - ERRMSG • 236
 - ERROR • 236
 - HILIGHT • 236
 - SEGKEY • 236
 - SEGMENT • 236
- SEGEXIT parameter • 446
 - IMSMFS Statement • 446
- SEGKEY parameter • 464
 - SEGEDIT Statement • 464
- SEGLOOP definition • 68, 129, 132, 134, 135, 137, 138, 139, 140, 141
 - creating a panel definition • 68, 129
- SEGLOOP page area • 405
 - calculating size • 405
- SEGLOOP processing • 61, 68, 71, 76, 77, 256
 - automatic line numbering • 256
 - browsing • 256
 - List screens • 256
 - paging • 256
 - Program definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
 - restrictions on FIELD statements • 256
 - restrictions with CICS BMS • 256
 - user-supplied processing • 256
- SEGLOOP processing, data positioning • 256
 - horizontal • 256
 - vertical • 256
- SEGLOOP processing, input processing • 256
 - parameters • 256
- SEGLOOP statement • 256
 - STBRKEY parameter • 256
- SEGLOOP statement, parameters • 167, 212, 224, 256, 405, 407
 - CINCRE • 256
 - ICUSTOM • 167, 212, 224
 - INCRE • 256
 - LINECNT • 256
 - OCUST1 • 224
 - OCUST2 • 224
 - OCUST3 • 224
 - PAGE • 256, 405
 - PAGESAV • 256, 405
 - SAVEKEY • 256, 405
 - SCHFLDC • 256, 405
 - SCHFLDI • 256, 405
 - SCHFLDL • 256, 405
- SEGLTH parameter • 464
 - SEGEDIT Statement • 464
- SEGMENT statement • 405
 - BROWSE • 405
- SEGMENT statement, parameters • 236, 405
 - KEYLEN • 405
 - WHEN • 236
- SEGNAME parameter • 464
 - SEGEDIT Statement • 464
- select • 115, 118, 119
 - field type • 115
- SELECT fields • 281, 340
 - determining where to pass control • 281
 - in prototyping • 340
- Select New Row Name (S137) • 640, 643, 650, 651, 653
 - declaring Global Temporary Tables • 640
 - Select New Row Name (S137) • 640
- SELKEY parameter • 407
 - FIELD statement • 407
- session controls • 105, 106, 107
 - User Profile Maintenance menu • 105
- size constraints • 325, 326
 - for Presentation Store • 325
- SIZE parameter • 453, 458, 464
 - NONTERM statement • 453

- REPORT Statement • 458
- SCREEN Statement • 464
- SIZE1 parameter • 451, 452, 470
 - MATCHKEY statement • 451
 - MERGEGRP statement • 452
 - SORTKEY statement • 470
- SIZE2 parameter • 451, 452, 470
 - MATCHKEY statement • 451
 - MERGEGRP statement • 452
 - SORTKEY statement • 470
- SKIPAFT parameter • 443
 - GROUP Statement • 443
- SKIPBEF parameter • 176, 443
 - GROUP statement • 176, 443
- Sort • 179, 183, 184, 186, 187, 188, 189, 410, 469
 - Mainline • 179
 - statement • 410, 469
- SPACMPT parameter • 446
 - IMSPGM Statement • 446
- SPASIZE parameter • 444, 446
 - IMSDRV Statement • 444
 - IMSPGM Statement • 446
- SPCLEEDIT • 313
 - in prototyping • 313
- SRC • 61, 68, 71, 76, 77
 - Program definition structure • 61, 68, 71
 - Program definition structure, batch • 76, 77
- SRC statement • 236
 - example • 236
 - in loop processing • 236
- SRTORDER parameter • 470
 - SORTKEY statement • 470
- SSALIST parameter • 464
 - SEGEDIT Statement • 464
- standard attributes • 250
 - ATTRINT • 250
 - ATTRPRO • 250
 - overriding • 250
- Standard Sort • 419
 - BATCH statement, STRUCTRE parameter • 419
- START parameter • 470, 471, 476
 - SORTKEY statement • 470
- STARTIO parameter • 449
 - JOURNAL statement • 449
- STBRKEY parameter • 466
 - SEGLOOP Statement • 466
- STORAGE parameter • 456, 469, 470

- PLIXOPT Statement • 456
- SORT statement • 469
- STRUCTRE parameter • 419
 - BATCH statement • 419
- SUMMARY • 73, 74
 - Batch Panel Definition • 74
 - definition of • 73
- Support • 633
 - DB2 Get Diagnostics • 633
- SVCODE parameter • 442
 - GETDIAG Statement • 442
- SYNONYM parameter • 432, 433, 476
 - DB2 Statement • 432
 - TABLE Statement • 476
- SYSID parameter • 423, 457, 458
 - CQUEUE statement • 423
 - RECORD statement • 457
- SYSMSG parameter • 446
 - IMSMFS Statement • 446
- System Development Life Cycle • 50, 52, 53
 - alterations to • 50

T

- TBLNAME parameter • 432, 459
 - DB2 Statement • 432
 - ROW Statement • 459
- TBLQUAL parameter • 432, 459
 - DB2 Statement • 432
 - ROW Statement • 459
- TDF screen organization • 105, 313
 - Prototyping Facility • 313
 - User Profile Maintenance • 105
- TDSKIP parameter • 443, 444
 - GROUP Statement • 443
- Temporary Storage • 412, 423
 - CQUEUE statement, QUETYPE parameter • 423
 - generating • 412
- TERM parameter • 174, 176, 437, 453
 - BATCH statement • 174
 - DRIVER Statement • 437
 - NONTERM statement • 453
- test data bases • 400, 401
 - allocating • 400
- TGREATR parameter • 450
 - MATCH statement • 450
- TLNNAME parameter • 476
 - TABLE Statement • 476
- TMPCOMT parameter • 458, 459

- REPORT Statement • 458
- TMPCUST parameter • 458
 - REPORT Statement • 458
- TMPNAME parameter • 458
 - REPORT Statement • 458
- TMPQUAL parameter • 458
 - REPORT Statement • 458
- TOPDTL • 73, 74
 - Batch Panel Definition • 74
 - definition of • 73
- TOPPAGE • 73, 74
 - Batch Panel Definition • 74
 - definition of • 73
- TPISIZE parameter • 444
 - IMSDRV Statement • 444
- TPOSIZE parameter • 444
 - IMSDRV Statement • 444
- TRACE parameter • 444, 446
 - IMSDRV Statement • 444
 - IMSPGM Statement • 446
- TRANCDE parameter • 444, 446
 - IMSDRV Statement • 444
 - IMSMFS Statement • 446
 - IMSPGM Statement • 446
- TRANFLD parameter • 446
 - IMSMFS Statement • 446
 - IMSPGM Statement • 446
- TRANKEY parameter • 451
 - MATCHKEY statement • 451
- TRANMFS parameter • 446
 - IMSMFS Statement • 446
- Transfer to TDF option • 318
 - in prototyping • 318
- Transfer Work Area • 256, 404, 405, 407, 408
 - contents • 404
 - in HELP/HOLD processing • 407
 - paging requirements • 256
 - save-key table • 256
 - SEGLOOP page area • 405
- transferring control • 253, 256
 - unconditionally • 253
 - using CONSYS • 253
 - using PFKEY • 253
 - using SCONSYS • 253
- Transient Data • 412, 423
 - CQUEUE statement, QUETYPE parameter • 423
 - generating • 412
- TSO Test PSB • 400

- creating • 400
- TYPE parameter • 433, 442, 466
 - DCLCOL Statement • 433
 - GETDIAG Statement • 442
 - SEGLOOP statement • 466

U

- Update Screen Params, screen • 250, 251
 - EATTR parameter • 250
- Update User-defined DATATYPES screen (S184)
 - 640
 - specified for SENCOLS • 640
- UPDTA parameter • 464
 - SCREEN Statement • 464
- UPRLWR parameter • 456
- USAGE parameter • 443, 459, 460, 466
 - GROUP Statement • 443
 - ROW Statement • 459
 - SEGLOOP Statement • 466
- User Exec • 61, 69, 71, 76, 77
 - creating a program definition • 69
 - creating a program definition, batch • 76
 - Program Definition structure • 61, 71
 - Program Definition structure, batch • 77
 - program structure • 61
- User Profile Maintenance • 98
 - TDF Main Menu • 98
- User Sort • 410, 411
 - data base storage • 410
 - generation of • 410
 - statement • 410
- USGCOPY parameter • 444, 446, 479, 480
 - IMSDRV Statement • 444
 - IMSPGM Statement • 446
 - TSOPGM Statement • 479
- Using alternate cursor • 636
 - FETCH Details Screen (S244) • 636
- Utilities • 98, 100
 - TDF Main menu • 98

V

- VALUES parameter • 230
 - example • 230
- FIELD statement • 230
- format • 230

W

- WAIT parameter • 449
 - JOURNAL statement • 449

WHEN parameter • 464
 SEGEDIT Statement • 464
WHERE parameter • 434, 459, 464
 DELETE Statement • 434
 ROW Statement • 459
 SEGEDIT Statement • 464
WKAREA parameter • 437, 453, 458, 464
 DRIVER Statement • 437
 NONTERM statement • 453
 REPORT Statement • 458
 SCREEN Statement • 464
WKSPAIN parameter • 444, 446
 IMSDRV Statement • 444
 IMSPGM Statement • 446
WKSPAIO parameter • 444, 446
 IMSDRV Statement • 444
 IMSPGM Statement • 446
WKSPASZ parameter • 444, 446, 448
 IMSDRV Statement • 444
 IMSPGM Statement • 446

X

XFEDIT • 61, 68, 71, 76, 77
 Program Definition structure • 61, 68, 71
 Program definition structure, batch • 76, 77
XFEDIT statement, parameters • 236
 COND • 236
 CURSOR • 236
 ERRCHAR • 236
 ERRMSG • 236
 HILIGHT • 236
XFER parameter • 437
 DRIVER Statement • 437
XFERWKA parameter • 437, 438, 439, 458, 464
 DRIVER Statement • 437
 REPORT Statement • 458
 SCREEN Statement • 464
XOPTS parameter • 456
 PLIXOPT Statement • 456