Geoff Nicholls
Paolo Bruni
Dougie Lawson
Egide Van Aershot

IMS 12: The IMS Catalog

Introduction

This document provides a description of the IBM® IMS™ catalog and the way it can be used to expand and consolidate the information about IMS databases and their metadata.

The IMS catalog is an optional system database, available with IMS 12, that stores metadata about your databases and applications. Its comprehensive view of IMS database metadata, fully managed by IMS, allows IMS to participate in solutions that require the exchange of metadata. A type of solution that requires such an exchange is business impact analysis.

The IMS Universal drivers are enhanced to take advantage of the IMS catalog.

In this paper, the following topics are described:

- Overview and objectives of the catalog
- Physical structure of the catalog database
- IMS catalog database installation and management
- Application use of the catalog
- The role of the IMS Enterprise Suite Explorer for Development
- Using IMS Explorer to capture IMS metadata
- Enhancements to the IMS Universal drivers
- Recommended maintenance

This IBM Redpaper™ publication describes the IMS catalog, a trusted online source for both IBM IMS database and application metadata information, and its use by IMS Open Database and IMS Explorer. This paper targets application developers who can benefit from these simplification and integration functions when they are facing any large-scale deployment of the IMS Open Database solution.

Overview and objectives of the catalog

Before the introduction of the IMS catalog, information about the structure of the Data Language/I (DL/I) databases was spread across the following different data structures:

- The database description

  The *database description* (DBD) defines the characteristics of a database. For example, the DBD defines characteristics such as its organization and access method, the segments and fields in a database record, and the relationship between the types of segments. Most of the time, information for the segments was limited to the few fields required to identify and search for segments across the hierarchical structures. That is, limited to the fields used as search arguments in the segment search argument (SSA), and the fields required to define logical relationships and secondary indexes.

- The COBOL copybooks and PL/I or C include members

  The details of the fields in each segment of the database are often defined in a Common Business Oriented Language (COBOL) copybook and in PL/I or C include members. These members detail all the fields in each database segment (not just the fields used in SSAs), and are included into the source program when the program is compiled.

- The program specification block

  A *program specification block* (PSB) is used to define two things: The view of the databases to be used by a program, and any logical message destinations. These views are called *program communication blocks* (PCBs), because they are the means of communicating between the application program and IMS. There can be many PCBs in a PSB (as shown in Figure 1), allowing a program to communicate with (access) multiple IMS databases. For database PCBs, the segments and the hierarchical structure that the program can access, are described. They can also indicate what sensitivity a program has to the information. That is, whether the program can see only a subset of the segment types in the database, and a subset of the fields in these segments.

  A PCB can also allow a program to use different access paths through a database. It can allow the program to access a database through a secondary index or a logical relationship. The program view of the hierarchical structure of the database can be different from the hierarchical structure defined in the DBD. See Figure 1.

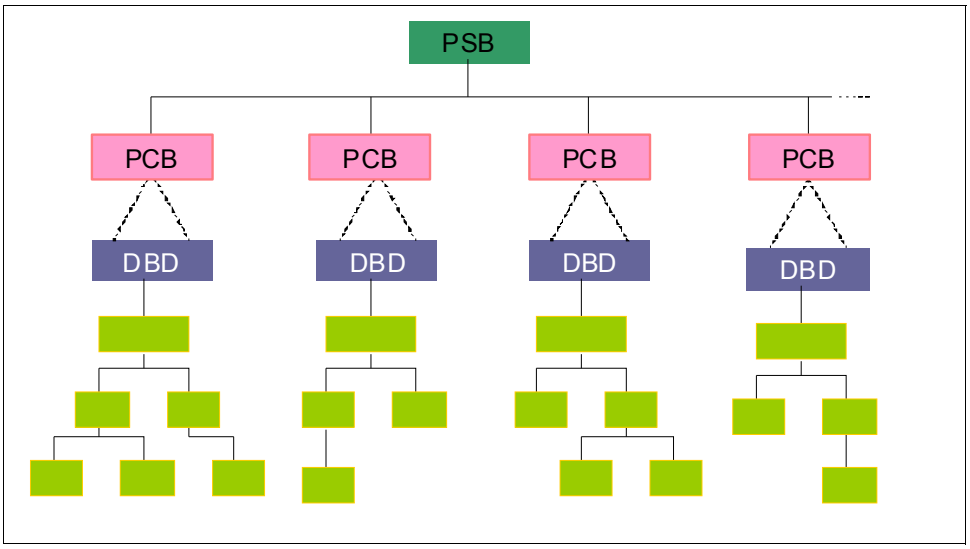


Figure 1 A PSB with multiple PCBs

For online IMS systems, *application control block* (ACB) libraries contain the metadata used by IMS, created by performing an ACB generation from the DBD and PSB libraries. The current system that uses DBD, PSB, and ACB libraries is proprietary. Typically, a DBD just defines a subset of the fields in a database record, such as the sequence (key) fields and any that are needed by a secondary index. Other fields in a record are only defined in a COBOL copybook or in a PL/I or C include file in the application program.

The introduction of Java access to IMS data from JBP or JMP regions, or from remote systems or clients, required easily accessible metadata. To provide this access, the contents of ACBLIB are offered as a Java class.

With the move to making IMS data more widely and easily accessible outside of the mainframe, application programmers need to have easy and consistent access to the metadata. Since IMS V8, this data was provided by using the DLIMODEL utility. This utility generates Java class files that contain a static definition of the database design at the point when the DLIMODEL utility is run.

The drawback when using the DLIMODEL utility is that it can be a challenge to manage. Also, change control is needed when the underlying database definition changes. This method potentially allows the creation of multiple copies of the database metadata, each one of which must be updated with any database structure change.

There is a requirement for a source of metadata that is easier to manage and can be trusted to reflect the possible changes of the design of IMS databases. The metadata can also be defined in an open format.

IMS 12 introduces the IMS catalog. The IMS catalog holds the metadata for databases and PSBs. It is accessed by using the Java Database Connectivity (JDBC) drivers and is also available to any tool or application. IMS metadata is available in an Extensible Markup Language (XML) format, and also available to standard DL/I applications in a traditional IMS segment format.

When the metadata is updated, the catalog is also updated to reflect the change; therefore, the data is always current. The data is current because the consumers get metadata dynamically from the active IMS catalog *high availability large database* (HALDB), rather than from static class files.

The catalog in IMS 12 also includes a versioning system. This system allows the current version of the metadata, and a user-specified number of previous versions, to be kept and available for the applications.

The IMS catalog is a *partitioned hierarchical indexed direct access method* (PHIDAM) database that contains trusted metadata for IMS databases and applications. The IMS Catalog Populate utility (DFS3PU00) can optionally be used to initially populate the catalog by reading an existing ACBLIB and loading the available metadata into the catalog. All the information that is contained in the runtime ACBLIB (which was derived from the DBDs and PSBs) is available to the users in the IMS catalog. Enhancements to the existing DBDGEN, PSBGEN, and ACBGEN processes allow database and application metadata to be defined to IMS. The new *ACB Generation and Catalog Populate* utility (DFS3UACB) automatically updates the IMS catalog when the ACB members are generated. This update keeps the catalog in a trusted state at all times after initialization. The IMS catalog is the single, authoritative source of database and application metadata for all client applications. The catalog can also contain application metadata such as decimal data specifications (scale and precision), data structure definitions, and mapping information.

The IMS catalog uses time stamps to identify the version of the metadata for each database, and can store multiple versions of metadata for each database. IMS provides the facility to keep a number of generations of this metadata, and remove old metadata, according to a maximum number of generations or longevity that you specify. By default, the IMS catalog contains information for each DBD and PSB in the active IMS system ACBLIB. The *IMS Catalog Record Purge* utility (DFS3PU10) must be run to remove the extra definitions in the catalog.

Physical structure of the catalog database

The IMS catalog is a HALDB database: a partitioned, IMS full-function database type. Before loading records into an IMS catalog, you must define the partitions to IMS. The IMS catalog is composed of the following objects:

- Primary database DFSCD000

  The access method is *overflow sequential access method* (OSAM), comprising the following database data sets:

  - Primary index data set
  - Indirect list data set (ILDS)
  - Four data set groups for the segments of the IMS catalog records

- Secondary index DFSCX000

  This index provides a cross-reference between the PSB segment and the DBD it is used to access: The data set for the secondary index database.

The data stored in the IMS catalog includes all the metadata that is traditionally held in the DBD and PSB libraries. It also includes additional information that is enabled by the catalog. The IMS Explorer for Development can also be used to expand the metadata stored in the catalog. See “The role of the IMS Enterprise Suite Explorer for Development” on page 32.

Segments of the catalog database

The following segments are featured in the catalog:

- Resource header (item name and type) for each DBD and PSB
- DBD resources:
  - Database structure definitions (ACCESS, RMNAME, and so on)
  - Data capture parameters
  - Physical database data set (DATASET) or area (AREA) definitions
  - Segment definitions (SEGM)
  - Field definitions (FIELD)
  - Marshaller definitions (DFSMARSH)
  - Logical children (LCHILD)
  - Indexed field (XDFLD)
  - Map definitions (DFSMAP)
  - Case definitions (DFSCASE)
  - Case field definition
  - Case marshaller definition
- PSB resources:
  - Program Communication Block (PCB)
  - Sensitive segments (SENSEG)
  - Sensitive fields (SENFLD)
  - DBD cross-reference

There are a number of new macros in a DBD which allow IMS 12 to better map the application data structures. They are all an extension and a subset of the field macro.

For each macro used in a DBD or PSB, there is an equivalent segment in the IMS catalog database.

The segments in the DBD of the IMS catalog database are shown in Example 1.

Example 1 Segments in the IMS catalog database (DFSCD000)

```
                                    DDDD  FFFFF  SSS   CCC  DDDD   000   000   000
                                    D   D F     S   S C   C D   D 0   0 0   0 0   0
                                    D   D F     S     C     D   D 0  00 0  00 0  00
                                    D   D FFF    SSS  C     D   D 0 0 0 0 0 0 0 0 0
                                    D   D F         S C     D   D 00  0 00  0 00  0
                                    D   D F     S   S C   C D   D 0   0 0   0 0   0
                                    DDDD  F      SSS   CCC  DDDD   000   000   000

                    VERSION=04/18/12  11.49
                    LEVELS=   8         SEGMENTS=  62       DATA SET GROUPS=   4
 ID=   1* N/A       PRIME DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           SEGMENT LENGTH MAX=   614, MIN=   126
                    OFLW  DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           KEY LENGTH MAX=        17, MIN=    16   NUMBSEG=   3
 ID=   2+ N/A       PRIME DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           SEGMENT LENGTH MAX=   934, MIN=    66
                    OFLW  DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           KEY LENGTH MAX=         2, MIN=     2   NUMBSEG=  11
 ID=   3" N/A       PRIME DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           SEGMENT LENGTH MAX=   934, MIN=   326
                    OFLW  DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           KEY LENGTH MAX=         2, MIN=     2   NUMBSEG=   3
 ID=   4. N/A       PRIME DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           SEGMENT LENGTH MAX=  4022, MIN=    46
                    OFLW  DS LOG. RCD. LEN=      0, BLOCKSIZE=  32752           KEY LENGTH MAX=        17, MIN=     2   NUMBSEG=  45
                                    C P  P L  L P   P  P  L  L E  RULES  PHYS.       SEG-NAME  D-B-NAME  FORM  LOG CHLD
SEG-NAME  SC# LV PAR -LEN- ---FREQ--- T T  P T  P H   C  C  C  C P        N-SEQ         OR        OR     OR   INSRT
                                    R FB   FB   FB .F .L .F .L S  I D R  INSRT       FLD-NAME  LEN STRT  PNTR  RULES
HEADER  *   1  1   0    56       0.00   XX          13             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  70 MAX=    56 PFX+MAX=    126
                                                                                                   MIN=    24 PFX+MIN=     94
DBD     *   2  2   1   552       0.00   XX X        10             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  62 MAX=   552 PFX+MAX=    614
                                                                                                   MIN=   505 PFX+MIN=    567
CAPXDBD .   3  3   2    32       0.00   X  X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  18 MAX=    32 PFX+MAX=     50
                                                                                                   MIN=    28 PFX+MIN=     46
DBDRMK  .   4  3   2   264       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                                   MIN=    20 PFX+MIN=     42
DSET    .   5  3   2    96       0.00   XX X         1             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  26 MAX=    96 PFX+MAX=    122
                                                                                                   MIN=    70 PFX+MIN=     96
DSETRMK .   6  4   5   264       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                                   MIN=    20 PFX+MIN=     42
AREA    .   7  3   2    40       0.00   XX X         1             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  26 MAX=    40 PFX+MAX=     66
                                                                                                   MIN=    26 PFX+MIN=     52
AREARMK .   8  4   7   264       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                                   MIN=    20 PFX+MIN=     42
SEGM    +   9  3   2   376       0.00   XX X         5             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  42 MAX=   376 PFX+MAX=    418
                                                                                                   MIN=   341 PFX+MIN=    383
CAPXSEGM.  10  4   9    32       0.00   X  X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  18 MAX=    32 PFX+MAX=     50
                                                                                                   MIN=    28 PFX+MIN=     46
SEGMRMK .  11  4   9   264       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                                   MIN=    20 PFX+MIN=     42
FLD     "  12  4   9   904       0.00   XX X         2             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  30 MAX=   904 PFX+MAX=    934
                                                                                                   MIN=   836 PFX+MIN=    866
FLDRMK  .  13  5  12   264       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                                   MIN=    20 PFX+MIN=     42
MAR     "  14  5  12   704       0.00   XX X         2             P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  30 MAX=   704 PFX+MAX=    734
                                                                                                   MIN=   640 PFX+MIN=    670
MARRMK  .  15  6  14   264       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                                   MIN=    20 PFX+MIN=     42
PROP    "  16  6  14   304       0.00   XX X                       P P P  LAST                              VAR LEN
                                                                                                *PFX*  LEN=  22 MAX=   304 PFX+MAX=    326
```

```
                                                                                          MIN=   264 PFX+MIN=    286
LCHILD  + 17  4   9    72      0.00   XX X         3             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  34 MAX=    72 PFX+MAX=    106
                                                                                          MIN=    56 PFX+MIN=     90
LCHRMK  . 18  5  17   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
LCH2IDX . 19  5  17    24      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=    24 PFX+MAX=     46
                                                                                          MIN=    20 PFX+MIN=     42
XDFLD   + 20  5  17   200      0.00   XX X         1             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  26 MAX=   200 PFX+MAX=    226
                                                                                          MIN=   184 PFX+MIN=    210
XDFLDRMK. 21  6  20   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
MAP     + 22  4   9   520      0.00   XX X         2             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  30 MAX=   520 PFX+MAX=    550
                                                                                          MIN=   264 PFX+MIN=    294
MAPRMK  . 23  5  22   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
CASE    + 24  5  22   656      0.00   XX X         2             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  30 MAX=   656 PFX+MAX=    686
                                                                                          MIN=   400 PFX+MIN=    430
CASERMK . 25  6  24   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
CFLD    + 26  6  24   904      0.00   XX X         2             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  30 MAX=   904 PFX+MAX=    934
                                                                                          MIN=   836 PFX+MIN=    866
CFLDRMK . 27  7  26   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
CMAR    + 28  7  26   704      0.00   XX X         2             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  30 MAX=   704 PFX+MAX=    734
                                                                                          MIN=   640 PFX+MIN=    670
CMARRMK . 29  8  28   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
CPROP   + 30  8  28   304      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   304 PFX+MAX=    326
                                                                                          MIN=   264 PFX+MIN=    286
DBDVEND . 31  3   2  4000      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=  4000 PFX+MAX=   4022
                                                                                          MIN=    64 PFX+MIN=     86
DBDSXXX . 32  3   2   135      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                                          MIN=    32 PFX+MIN=     54
DBDPXXX . 33  3   2   135      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                                          MIN=    32 PFX+MIN=     54
DBDRES1 . 34  3   2   135      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                                          MIN=    32 PFX+MIN=     54
DBDRES2 . 35  3   2   135      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                                          MIN=    32 PFX+MIN=     54
DBDHXXX . 36  2   1   128      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                                          MIN=    32 PFX+MIN=     54
PSB     * 37  2   1    88      0.00   XX X         7             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  50 MAX=    88 PFX+MAX=    138
                                                                                          MIN=    72 PFX+MIN=    122
PSBRMK  . 38  3  37   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
PCB     + 39  3  37   288      0.00   XX X         2             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  30 MAX=   288 PFX+MAX=    318
                                                                                          MIN=   208 PFX+MIN=    238
PCBRMK  . 40  4  39   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
SS      + 41  4  39   328      0.00   XX X         2             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  30 MAX=   328 PFX+MAX=    358
                                                                                          MIN=   320 PFX+MIN=    350
SSRMK   . 42  5  41   264      0.00   XX X                       P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                                          MIN=    20 PFX+MIN=     42
SF      + 43  5  41    40      0.00   XX X         1             P P P  LAST                         VAR LEN
                                                                            *PFX*  LEN=  26 MAX=    40 PFX+MAX=     66
                                                                                          MIN=    20 PFX+MIN=     46
SFRMK   . 44  6  43   264      0.00   XX X                       P P P  LAST                         VAR LEN
```

```
                                                                  *PFX*  LEN=  22 MAX=   264 PFX+MAX=    286
                                                                         MIN=    20 PFX+MIN=     42
DBDXREF .  45  3  37    48       0.00   XX X                    P P P  LAST   SNDIXD              VAR LEN
                                                                  *PFX*  LEN=  22 MAX=    48 PFX+MAX=     70
                                                                         MIN=    36 PFX+MIN=     58
                                                                  *LC**  DBDPSB    DFSCX000  INDX  LAST
                                                                  *XFD*  DBD2PSB  SECONDARY INDEXED FIELD
                                                               SUBSEQUENCE  **SRCSEG** **SAME**
                                                                  **SEARCH** IMSNAME    8   29
                                                                             PSBNAME    8   37
                                                                             TSVERS    13   13
                                                                  **SUBSEQ** /SX1       8    1
PSBVEND .  46  3  37  4000       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=  4000 PFX+MAX=   4022
                                                                         MIN=    64 PFX+MIN=     86
PSBSXXX .  47  3  37   135       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                         MIN=    32 PFX+MIN=     54
PSBRES1 .  48  3  37   135       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                         MIN=    32 PFX+MIN=     54
PSBRES2 .  49  3  37   135       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   135 PFX+MAX=    157
                                                                         MIN=    32 PFX+MIN=     54
PSBHXXX .  50  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
DBEXXXX .  51  2   1   135       0.00   XX X          1         P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  26 MAX=   135 PFX+MAX=    161
                                                                         MIN=    32 PFX+MIN=     58
DBESXXX .  52  3  51  4000       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=  4000 PFX+MAX=   4022
                                                                         MIN=    64 PFX+MIN=     86
UXX     .  53  2   1   128       0.00   XX X          1         P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  26 MAX=   128 PFX+MAX=    154
                                                                         MIN=    32 PFX+MIN=     58
UXXOXXX .  54  3  53   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
UXXHXXX .  55  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
SXX     .  56  2   1   128       0.00   XX X          1         P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  26 MAX=   128 PFX+MAX=    154
                                                                         MIN=    32 PFX+MIN=     58
SXXOXXX .  57  3  56   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
SXXHXXX .  58  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
RESERVE1.  59  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
RESERVE2.  60  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
RESERVE3.  61  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
RESERVE4.  62  2   1   128       0.00   XX X                    P P P  LAST                VAR LEN
                                                                  *PFX*  LEN=  22 MAX=   128 PFX+MAX=    150
                                                                         MIN=    32 PFX+MIN=     54
```

The hierarchical structure of the IMS catalog database is shown in Figure 2 on page 8. Some segment types were omitted for display reasons, but most of the important ones are shown. Example 1 on page 5 shows the complete list. To see the complete picture of the catalog database, import the catalog database DBD (DFSCD001) into IMS Explorer. The source of this DBD is available in the SDFSSRC data set delivered with IMS V12. Instructions for importing DBDs into IMS Explorer are in “Using IMS Explorer to capture IMS metadata” on page 35.

Figure 2 on page 8 shows the database structure for the IMS catalog database.

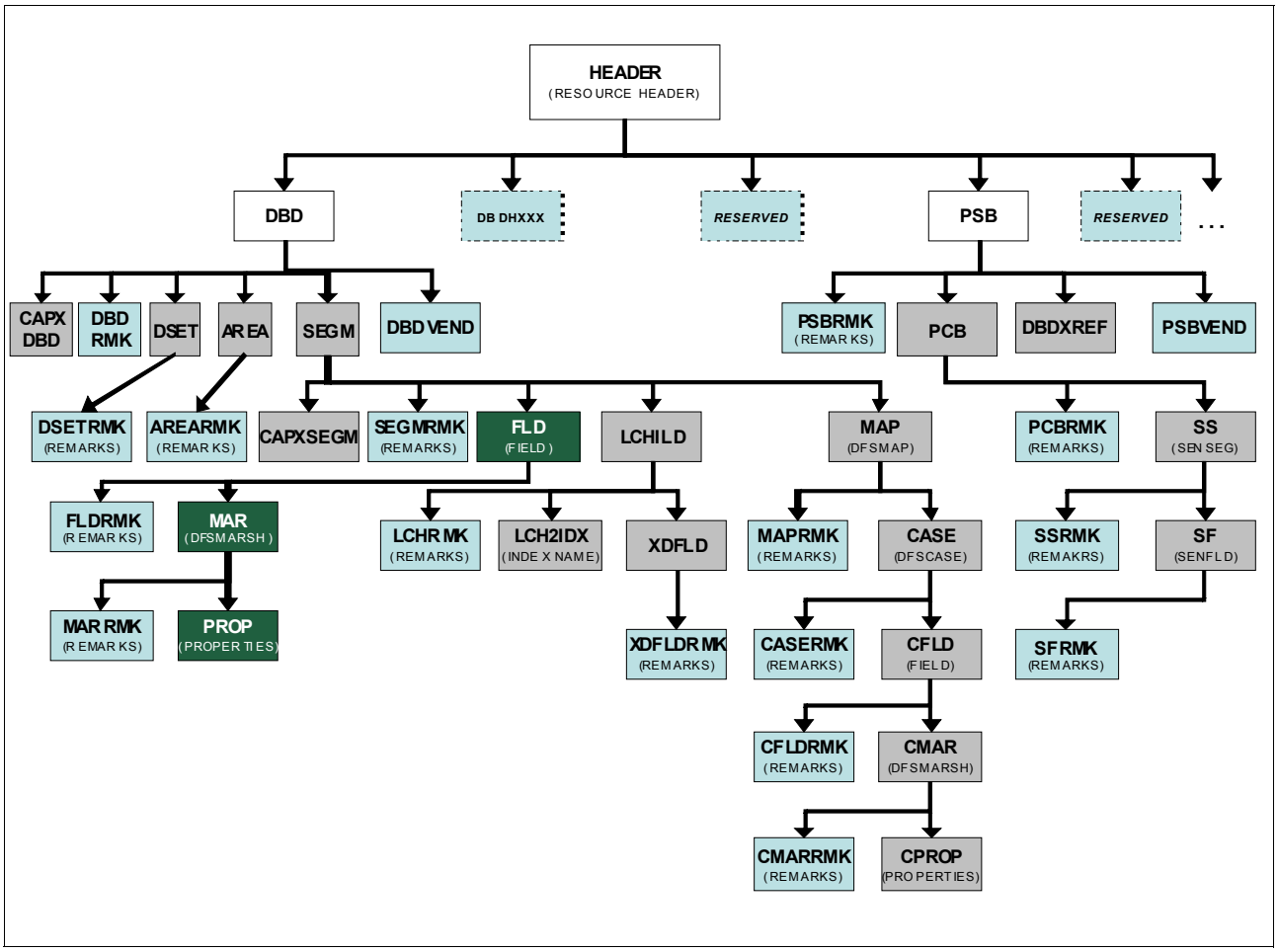


Figure 2 Database structure for the catalog database, DFSCD000

There are five PSBs supplied for access and update to the catalog database:

- DFSCPL00

  It is used for an initial load of the database (PROCOPT=L).

- DFSCP000

  It has read-only access (PROCOPT=G) for all segments. This PSB can be used for Assembler and COBOL programs for direct read-only access to the catalog database.

- DFSCP001

  It has update access (PROCOPT=A) for all segments. This PSB can be used for Assembler and COBOL programs for direct update access to the catalog database.

- DFSCP002

  It has read-only access (PROCOPT=G) for all segments. This PSB is used for PL/I programs for direct access to the catalog database.

- DFSCP003

  It has read-only access (PROCOPT=G) for all segments. This PSB is used for PASCAL programs for direct access to the catalog database.

There are several segments in the catalog database, including DBDVEND and PSBVEND, in the database definition for IBM tools and vendor products to store additional information. There are also several reserved segments in the database to allow for future expansion.

Most of the segment types (such as DBD, DSET, AREA, SEGM, FLD, LCHILD, XFLD) contain metadata about the specific element that they represent. The data also includes the name of the segment corresponding to the metadata stored.

Other segments (DBDRMK, DSETRMK, AREARMK, SEGMRMK, MAPRMK, CASERMK) contain remarks about the element definition.

The CAPXDBD and CAPXSEGM segments contain information about Data Capture exit routines used.

IMS catalog database installation and management

The IMS catalog is implemented as a HALDB database. IMS normally requires each HALDB database to be registered in the Database Recovery Control (DBRC). There are companies who choose not to use DBRC in some of their environments. Therefore, a method is provided to define the IMS catalog without registering the database in the RECON.

In this section, we describe the following steps for installation and management of the IMS catalog:

- Installation
- IMS catalog initial data population
- ACB generation and changes
- IMS Catalog Copy utility
- Using the IMS catalog without DBRC
- Automatically creating the IMS catalog database data sets
- Keeping multiple versions of metadata in the catalog
- Aliases and sharing
- Definitions needed for the IMS catalog

IMS provides a number of new utilities to maintain the catalog database when databases or PSB definitions are changed as part of your normal application development lifecycle.

Installation

The first step of the installation of the IMS catalog is to copy the supplied catalog DBD and PSB members from the SDFSRESL data set to your own DBD and PSB libraries. See Example 2.

The source for the catalog DBD and PSBs is shipped for reference in the IMS source library SDFSSRC. However, the object code in SDFSRESL must be used for execution.

Example 2 Copy the supplied DBD and PSB members to your own libraries

```
//CPYCMEM EXEC PGM=IEBCOPY
//SDFSRESL DD  DSN=IMS12.SDFSRESL,DISP=SHR
//DBDLIB   DD  DSN=IMS12.IMS12X.DBDLIB,DISP=SHR
//PSBLIB   DD  DSN=IMS12.IMS12X.PSBLIB,DISP=SHR
//SYSIN    DD  *
          COPY   OUTDD=DBDLIB,INDD=((SDFSRESL,R))
          SELECT MEMBER=(DFSCD000,DFSCX000)
          COPY   OUTDD=PSBLIB,INDD=((SDFSRESL,R))
          SELECT MEMBER=(DFSCPL00,DFSCP000,DFSCP001,DFSCP002,DFSCP003)
```

After the DBDs and PSBs are copied to your libraries, you need to build them as ACBs by using the traditional ACB process. See Example 3.

Example 3 ACBGEN for IMS catalog

```
// JCLLIB ORDER=IMS12.PROCLIB
// EXEC ACBGEN
//SYSIN    DD  *
    BUILD PSB=(DFSCPL00)
    BUILD PSB=(DFSCP001)
    BUILD PSB=(DFSCP000)
    BUILD PSB=(DFSCP002)
    BUILD PSB=(DFSCP003)
```

Next, create the catalog database data sets. The space allocated for the catalog database needs to be sufficient to store the number of DBD and PSB definitions for your environment. This creation can be done in three ways:

- The database data sets can be allocated with the *job control language* (JCL) similar to what is shown in Example 4.
- The *IMS Catalog Partition Definition Data Set* utility (DFS3UCD0) can be used to create the catalog partition definition database data set if DBRC is not in use. It also creates the database data sets for the catalog database.
- The IMS Catalog Populate utility (DFS3PU00) is used to load or insert records into the catalog database, whether DBRC is used for the catalog database. If any of the database data sets do not exist, the utility creates them. The size parameters used for automatic creation are specified in the catalog section of the DFSDFxxx procedure library (PROCLIB) member.

Example 4 shows the process of defining the database data sets and indexes.

Example 4 Define the HALDB data sets, the ILDS, and the primary and secondary indexes

```
//PHIDAM EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
   ALLOCATE DSNAME('IMS12.IMS12X.DFSCDI2D.A00001') FILE(A00001) -
       RECFM(F,B,S) DSORG(PS) NEW CATALOG -
       SPACE(2,2) CYLINDERS VOLUME(SBOXI5) UNIT(SYSALLDA)

   ALLOCATE DSNAME('IMS12.IMS12X.DFSCDI2D.B00001') FILE(B00001) -
       RECFM(F,B,S) DSORG(PS) NEW CATALOG -
       SPACE(2,2) CYLINDERS VOLUME(SBOXI5) UNIT(SYSALLDA)

   ALLOCATE DSNAME('IMS12.IMS12X.DFSCDI2D.C00001') FILE(C00001) -
       RECFM(F,B,S) DSORG(PS) NEW CATALOG -
       SPACE(2,2) CYLINDERS VOLUME(SBOXI5) UNIT(SYSALLDA)

   ALLOCATE DSNAME('IMS12.IMS12X.DFSCDI2D.D00001') FILE(D00001) -
       RECFM(F,B,S) DSORG(PS) NEW CATALOG -
       SPACE(2,2) CYLINDERS VOLUME(SBOXI5) UNIT(SYSALLDA)

   DEFINE CLUSTER(NAME('IMS12.IMS12X.DFSCDI2D.L00001') -
      FREESPACE(80 10) SHAREOPTIONS(3 3) KEYS(9 0) -
      RECORDSIZE(50 50) SPEED CYLINDERS(1,1) VOLUMES(SBOXI5)) -
      DATA(CONTROLINTERVALSIZE(4096)) -
      INDEX(CONTROLINTERVALSIZE(2048))

   DEFINE CLUSTER(NAME('IMS12.IMS12X.DFSCDI2D.X00001') INDEXED -
     KEYS(16,5) VOL(SBOXI5) REUSE RECORDSIZE (22,22)) -
```

```
      DATA(CONTROLINTERVALSIZE(4096))

//PSINDEX EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
  DEFINE CLUSTER (NAME('IMS12Q.IMS12X.DFSCXI2D.A00001') INDEXED -
     SHAREOPTIONS(3 3) KEYS(37,45) REUSE RECORDS(5,5) VOL(SBOXI5) -
     RECORDSIZE(82,82)) DATA(CONTROLINTERVALSIZE(4096))
```

After the database data sets are created, you need to update the IMS.PROCLIB(DFSDFxxx) member to define the catalog parameters for your IMS system. There are two sections to be added to the member, depending on which of the following configurations you choose to use:

- A unique IMS catalog for each system.
- A unique DFSDFxxx member for each system.
- A shared IMS catalog.
- A shared DFSDFxxx member.

Example 5 shows the simplest environment: a single IMS with a single catalog prefixed with the default name DFSC.

Example 5 IMS.PROCLIB(DFSDFxxx) for a single IMS system

```
*-------------------------------------------------------------------*
* IMS CATALOG SECTION                                               *
*-------------------------------------------------------------------*
<SECTION=CATALOG>
  CATALOG=Y
  ALIAS=DFSC
```

Example 6 shows a shared DFSDFxxx member with a separate IMS catalog for each IMS system (defined with an ALIAS that matches the IMSID). We explain the use of aliases for the IMS catalog in “Aliases and sharing” on page 17.

Example 6 IMS.PROCLIB(DFSDFxxx) for multiple IMS systems

```
*-------------------------------------------------------------------*
* IMS CATALOG SECTION for IMS I12B *
*-------------------------------------------------------------------*
<SECTION=CATALOGI12B>
  CATALOG=Y
  ALIAS=I12B
*-------------------------------------------------------------------*
* IMS CATALOG SECTION for IMS I12D *
*-------------------------------------------------------------------*
<SECTION=CATALOGI12D>
  CATALOG=Y
  ALIAS=I12D
```

The catalog can be activated, by using the new DFSDFxxx member, with a cold start, warm start, or even an emergency restart of the IMS system. The restart is required because IMS only reads the DFSDF member during initialization.

IMS catalog initial data population

After you define the IMS catalog database data sets, run a new utility to read the IMS 12 ACBLIB to initially load (populate) the IMS catalog database. As is usual for all new releases of IMS, the ACBLIB needs to be rebuilt from the DBD and PSB libraries. This rebuild is done by using the type-1 ACB generation utility for the new release before the ACBLIB is used, including populating the catalog. The catalog populate process might need to be done only one time for each system by using that IMS catalog.

You start by defining the IMS catalog to DBRC. This process can be done with the batch DBCRC utility (DSPURX00), by using the INIT.DB and INIT.PART commands for the catalog database. Or, the catalog populate process can be done by using the IMS Partition Definition Utility (PDU) application in the Time Sharing Option (TSO). DBRC registration is optional. For more information, see “Using the IMS catalog without DBRC” on page 16.

The initial data population of the IMS catalog is done by using a new utility, the *IMS Catalog Populate* utility (DFS3PU00), as shown in Figure 3.

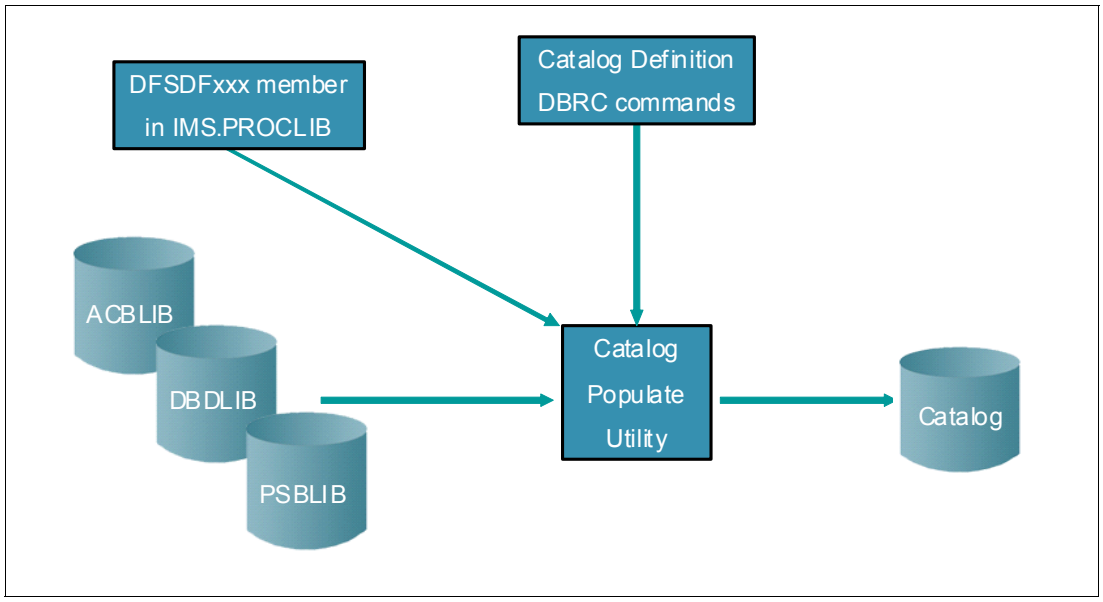


Figure 3 IMS Catalog Populate utility (DFS3PU00)

The utility reads the IMS.PROCLIB(DFSDFxxx) member, then builds the IMS catalog from the ACB libraries referenced in the JCL. The utility also reads the DBD and PSB libraries to capture the information about any generalized sequential access method (GSAM) DBDs. This method is used because they are not built as ACB library members.

After execution, a HALDB REORG record is recorded in the RECON.

The IMS Explorer allows us to add application metadata to these database definitions. This process is allowed because, at this stage, the IMS catalog holds only the rudimentary information that is available from the DBD and PSB definitions. For details about this process, see “The role of the IMS Enterprise Suite Explorer for Development” on page 32.

Example 7 on page 13 shows the JCL used to populate an IMS catalog from the current ACB libraries.

Example 7 Running the IMS Catalog Populate utility (DFS3PU00)

```
//LOADCAT EXEC PGM=DFS3PU00,
// PARM=(DLI,DFS3PU00,DFSCPL00,,,,,,,,,,Y,N,,,,,,,,,,,'DFSDF=12D')
//STEPLIB  DD  DSN=IMS12.SDFSRESL,DISP=SHR
//DFSRESLB DD  DSN=IMS12.SDFSRESL,DISP=SHR
//IMS      DD  DSN=IMS12.IMS12X.PSBLIB,DISP=SHR
//         DD  DSN=IMS12.IMS12X.DBDLIB,DISP=SHR
//PROCLIB  DD  DSN=IMS12.PROCLIB,DISP=SHR
//SYSABEND DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//IEFRDER  DD  DISP=(,CATLG),DSN=IMS12.IMS12X.PU000.LOG(+1),
// SPACE=(TRK,(5,5),RLSE),UNIT=SYSALLDA
//DFSVSAMP DD  DSN=IMS12.PROCLIB(DFSVSMDB),DISP=SHR
//IMSACB01 DD  DSN=IMS12.IMS12D.ACBLIBA,DISP=SHR
//IMSACB02 DD  DSN=IMS12.IMS12D.ACBLIB.DOPT,DISP=SHR
```

ACB generation and changes

After you migrate to an IMS system that is using an IMS catalog, you need to keep the ACB libraries and the catalog synchronized. IMS uses time stamps to ensure consistency. With time stamp, you can ensure that the catalog data can always be trusted as an accurate equivalent of the metadata that is held in the DBD, PSB, and ACB libraries.

There is a new *ACB Generation and IMS Catalog Populate utility* (DFS3UACB). This utility is used to perform both the generation of ACB members in an IMS.ACBLIB data set and the creation of the corresponding metadata records in the IMS catalog in a single job step. This process is shown in Figure 4. The DFS3UACB utility can be used in load mode to initially populate the catalog, or in update mode to add a version of the new or changed definitions. In update mode, a new version of any changed definitions are created in the catalog, rather than altering the existing definitions in the catalog.

When the utility is run in load mode, all existing records in the IMS catalog are discarded.

Figure 4 shows the ACB Generation and IMS Catalog Populate utility.

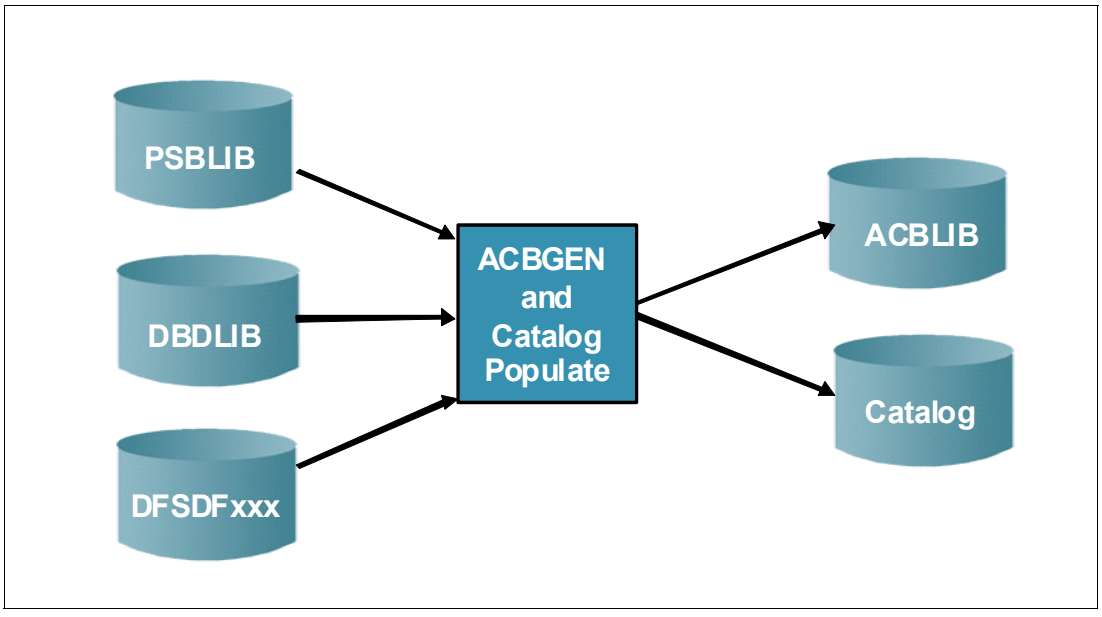


Figure 4 ACB Generation and IMS Catalog Populate utility

The new ACB Generation utility writes logs. If the IMS catalog is defined in DBRC, it updates the RECON. It can run as a bean-managed persistence (BMP) or DLIBATCH utility. If it is run as a BMP, ensure that the catalog database is opened by IMS for update access. Typically, IMS opens the catalog database in read-only mode.

IMS Catalog Copy utility

As soon as you migrate to an IMS catalog, ensure that the IMS catalog is updated every time you update an IMS ACB library. When you migrate applications from one environment to another, from the test through to production for example, the new *IMS Catalog Copy* utility can be used to keep the metadata synchronized. This utility allows the export and import of metadata information between catalogs:

- The utility DFS3CCE0 exports from a catalog and the ACBLIB, DBDLIB, or PSBLIB libraries.

  This utility copies an IMS catalog and any included ACB, DBD, and PSB libraries to export data sets in the same job step.

- The utility DFS3CCI0 imports from the export data set into another catalog, ACBLIB, DBDLIB, and PSBLIB.

  At the destination environment, the import module DFS3CCI0 loads or updates an IMS catalog. The module also copies any included ACB, DBD, and PSB libraries from the export data sets into their destination data sets.

  For the import function of the IMS Catalog Copy utility, the primary input to the utility is a data set that contains the new copy of the IMS catalog. A CCUCATIM data definition (DD) statement is required to identify this data set.

  If the ACB libraries, DBD libraries, and PSB libraries were copied during the export function, they are identified in the import JCL by the following DD statements:

  - CCUACBIM DD statement for the ACB library export data set
  - CCUDBDIM DD statement for the DBD library export data set
  - CCUPSBIM DD statement for the PSB library export data set

The new utilities ensure that the updates to the ACB libraries result in parallel updates to the IMS catalog.

Optionally, the IMS Catalog Copy utility can also create copies of the ACB, DBD, and PSB library members that correspond to the IMS catalog records that are copied.

The IMS Catalog Copy utility creates an import statistics report for the record segments to be loaded or updated in the IMS catalog. When the Catalog Copy utility runs in analysis-only mode, the report reflects only certain statistics. The report reflects only the potential statistics if the IMS catalog were loaded or updated from the ACB libraries that are currently used as input to the utility.

The IMS Catalog Copy utility can be run as either a batch job or as an online BMP.

Keeping multiple versions of metadata in the catalog

The catalog is designed to hold multiple versions of the DBD and PSB metadata. The information in the IMS catalog is time stamped. IMS uses these time stamps to allow multiple generations of the metadata to be kept.

The catalog section of the DFSDFxxx PROCLIB member specifies the IMS wide default values for the retention of metadata in the catalog. The parameters define the maximum number of generations and the minimum retention period for which information is to be kept in the catalog. This specification is similar to the way that information is stored by DBRC in the RECONs to keep records of a number of Image Copy sets. However, the information is not automatically purged, as it is with DBRC. The DBDs and PSBs are only deleted by the *IMS Catalog Record Purge* utility (DFS3PU10). The retention parameters for specific databases can be specified with DFS3PU10.

The following two parameters are used to control the optional RETENTION statement in the DFSDFxxx catalog section: VERSIONS=nnn and DAYS=ddd:

- VERSIONS defines the maximum number of versions of a DBD or PSB to be kept in the IMS catalog database. The value can be from 1 to 65535; the default value is 2. When the maximum value is reached, the oldest version (based on the ACBGEN time stamp) is replaced by the newest.
- DAYS defines the minimum number of days a version remains in the catalog. The value can be from 0 to 65535; the default value is 0 (function disabled). When a version of the catalog metadata is older that the specified version, it becomes a candidate for removal. It might be removed when new versions of the same DBD or PSB are added to the IMS catalog.

They work in the same way that GENMAX and RECOVPD work with DBRC: If the maximum number of versions is reached but the retention period is not expired, a new catalog record is kept. The oldest record is not removed.

Example 8 shows the specification in the IMS.PROCLIB(DFSDFxxx) member, where at least five generations of metadata is kept, for a minimum of 30 days.

Example 8 Setting default maximum generations and retention periods for catalog metadata

```
<SECTION=CATALOG>
CATALOG=Y
ALIAS=DFSC
RETENTION=(VERSIONS=5,DAYS=30)
```

With multiple versions stored in the catalog, if an online change is reversed, the catalog might still have the previous version available.

For logically related databases, where in the past the DBD information is kept only in the DBDLIB and not the ACBLIB, the Catalog Populate utility sets the time stamp to zero. The time stamp is set to zero until the combined ACBGEN/Populate utility runs for those databases.

Metadata information in the catalog can be removed by the IMS Catalog Record Purge utility (DFS3PU10) if it is outdated or if the version exceeds the specified retention value. When using DFS3PU10, the default of two versions applies. If no retention value is specified, each execution of ACBGEN or Populate Utility adds a version.

IMS Catalog Record Purge utility

The IMS Catalog Record Purge utility (DFS3PU10) provides several functions:

- Sets the record retention criteria for specific DBD and PSB records in the catalog database.
- Produces a list of DBDs and PSBs with versions that are no longer needed according to the current retention criteria for each record.
- Purges specific versions (based on the ACB time stamp) of a record for a DBD or PSB resource from the IMS catalog database.
- Purges unnecessary versions of IMS catalog records from the catalog database based on criteria that you specify.

Automatically creating the IMS catalog database data sets

Use the DFSDFxxx member of the IMS PROCLIB data set to specify processing options for the IMS catalog. The DFSDFxxx member contains parameters organized into sections. Example 9 shows the specify options for the IMS catalog. In a data sharing environment, this section defines the IMS catalog for all IMS systems that are not individually configured with a CATALOGxxxx section.

Example 9 IMS catalog parameters in DFSDFxxx

```
<SECTION=CATALOG>
CATALOG=Y
ALIAS=DFSC
/* IXVOLSER=vvvvvv if DFSMS is not used */
DATACLAS=IMSDATA
MGMTCLAS=EXTRABAK
STORCLAS=IMS
SPACEALLOC=500
```

The additional parameters allow us to define the DFSMS DATALCLAS, MGMTCLAS, STORCLAS, or volume serial number if DFSMS is not used.

Parameter SPACEALLOC=nnnn (with a range of 0 to 9999; the default is 500) also allows us to define the space as a percentage of the predefined IMS value.

Using the IMS catalog without DBRC

The IMS catalog database is a standard PHIDAM database. Usually, that requires registration to DBRC. DBRC tracks the logs used when the database is updated. DBRC can also be used to generate jobs to perform image copies and accumulate changes with the IMS Change Accumulation utility.

You might be running your IMS system with non-HALDB databases that are not registered to DBRC (which is a common configuration for test systems). If the databases are not registered, IMS does not insist that the catalog HALDB database (and index) is registered in RECON. If the HALDB is not registered, IMS does not track logs, change accumulations, or image copies.

The backup, logging, and recovery of the IMS catalog is the responsibility of the user. This process can be done by running stand-alone IBM MVS™ utilities or by DFSMShsm backup and recovery. In some cases, the use of the IMS Catalog Populate utility to rebuild the database might be the best choice.

To overcome the need for IMS to have the IMS catalog PHIDAM database metadata (normally stored in the DBRC RECON), IMS 12 has special handling for the IMS catalog. Instead of having database, partition, and database data set records defined in the RECON, IMS uses the DFSMDA macro with the TYPE=CATDBDEF statement. This statement defines the dynamic allocation parameter list for the IMS catalog partition definition data set. This data set contains the definitions for the catalog HALDBs that are not defined in the DBRC RECON data set. The DD name of the catalog partition definition data set is DFSHDBSC.

A new utility DFS3UCD0 is used to create the HALDB partition definition data set. You need to supply a system input stream (SYSIN) data set to define the data that you normally use on INIT.DB and INIT.PART, if the database was registered in DBRC. Example 10 shows an example. The structure information is stored in the definition data set (DFSHDBSC).

Example 10 Example SYSIN for DFS3UCD0

```
HALDB=(NAME=DFSCD000,HIKEY=YES)

PART=(NAME=DFSCD000,
    PART=DFSCD01,
    DSNPREFX=IMSTESTS.DFSCD000,
    KEYSTHEX=FFFFFFFFFFFFFFFFFFFFFFFFFFFF)

HALDB=(NAME=DFSCX000,HIKEY=YES)

PART=(NAME=DFSCX000,
    PART=DFSCX01,
    DSNPREFX(IMSTESTS.DFSCX000)
    KEYSTHEX=FFFFFFFFFFFFFFFFFFFFFFFFFFFF)
```

The next thing you need to add is a statement in the IMS.PROCLIB(DFSDFxxx) member to define the IMS catalog that is not registered in DBRC (Example 11).

Example 11 IMS.PROCLIB(DFSDFxxx) definition for a non-DBRC IMS catalog

```
<SECTION=DATABASE>
UNREGCATLG=DFSCD000
```

Optionally, to avoid making JCL changes, you can also build a DFSMDA dynamic allocation macro to define the DFSHDBSC to IMS, as shown in Example 12. TYPE=CATDBDEF is a new specification just for the unregistered IMS catalog database.

Example 12 DFSMDA definition

```
        DFSMDA TYPE=INITIAL
        DFSMDA TYPE=CATDBDEF,DSNAME=IMS12.IMS12D.CATALOG.DEFDS
        DFSMDA TYPE=FINAL
        END
```

Aliases and sharing

In a complex IMSplex, you have a number of options for how to set up the ACBLIB and the IMS catalog. You can share or clone the ACB libraries ACBLIBA or ACBLIBB. Similarly, the IMS catalog can be shared, or it can use a separate IMS catalog for each IMS system.

The way that ACB libraries in IMS 10 or IMS 11 are used depends on whether you are using local, global, and member online changes. IMS 12 supports these configuration options, and you can configure the IMS catalog to participate with your existing shared or discrete ACB libraries. The IMS catalog can be shared between multiple IMS systems.

Tip: Convert your systems to use global online change, which dynamically allocates ACB libraries and the use of member online change.

Figure 5 shows the migration of a system from an IMSplex where each IMS has its own cloned ACB library pair. However, we also introduce a single shared IMS catalog populated from all the existing ACB libraries.

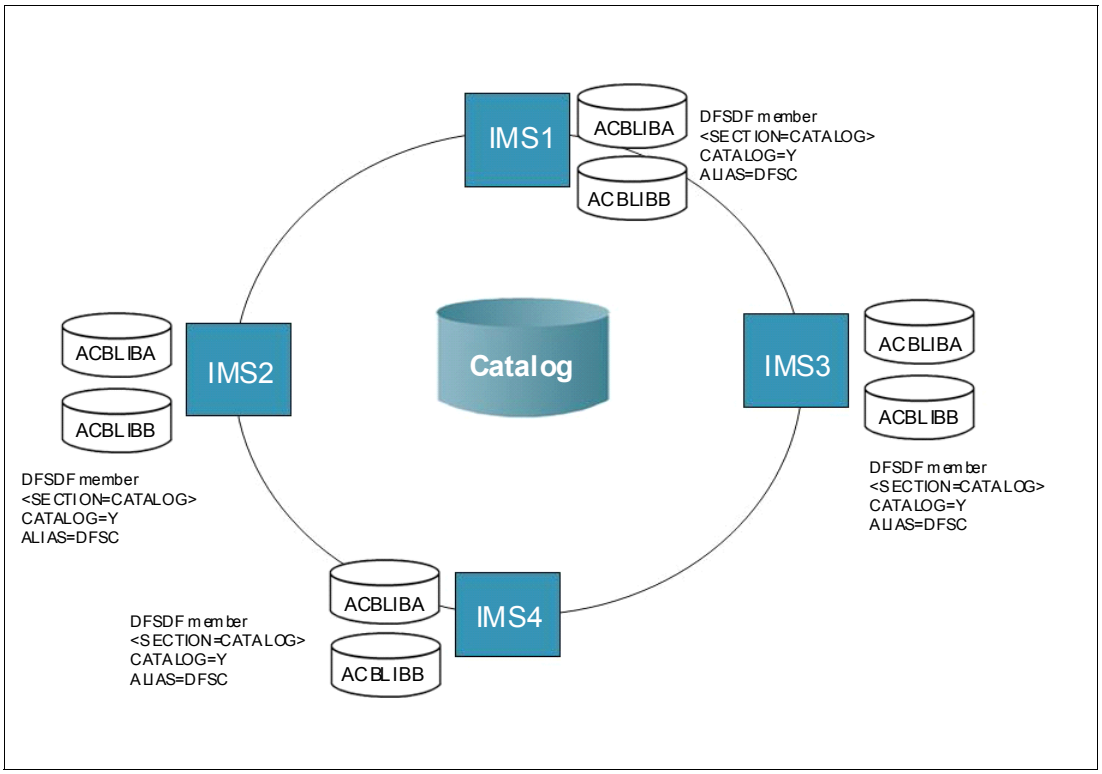


Figure 5 Multiple IMS, cloned ACBLIB, and shared IMS catalog

Figure 6 on page 19 shows a system where we are starting from separate (cloned) ACB libraries. We chose to create a unique aliased IMS catalog for each IMS system.

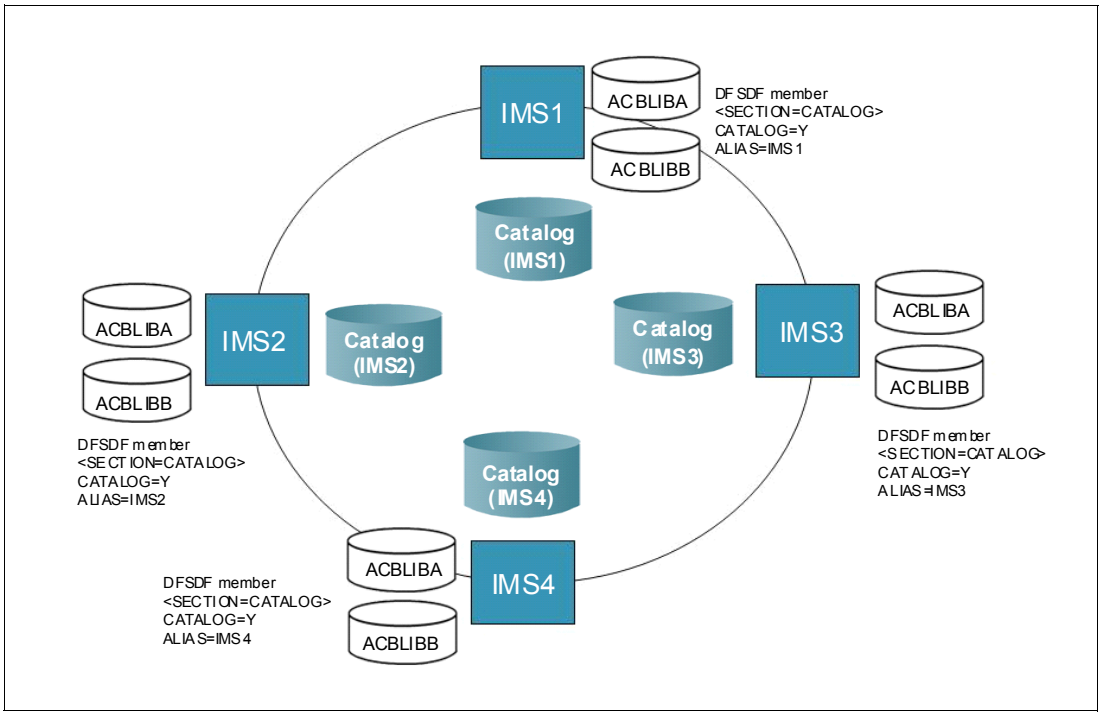


Figure 6 Multiple IMS, cloned ACBLIB, and one IMS catalog for each IMS

Figure 7 shows the integrated system, where there is one pair of ACB libraries for the IMSplex and a single shared IMS catalog (registered in DBRC with SHARELVL=3). In this system, global online change and member online change are used.

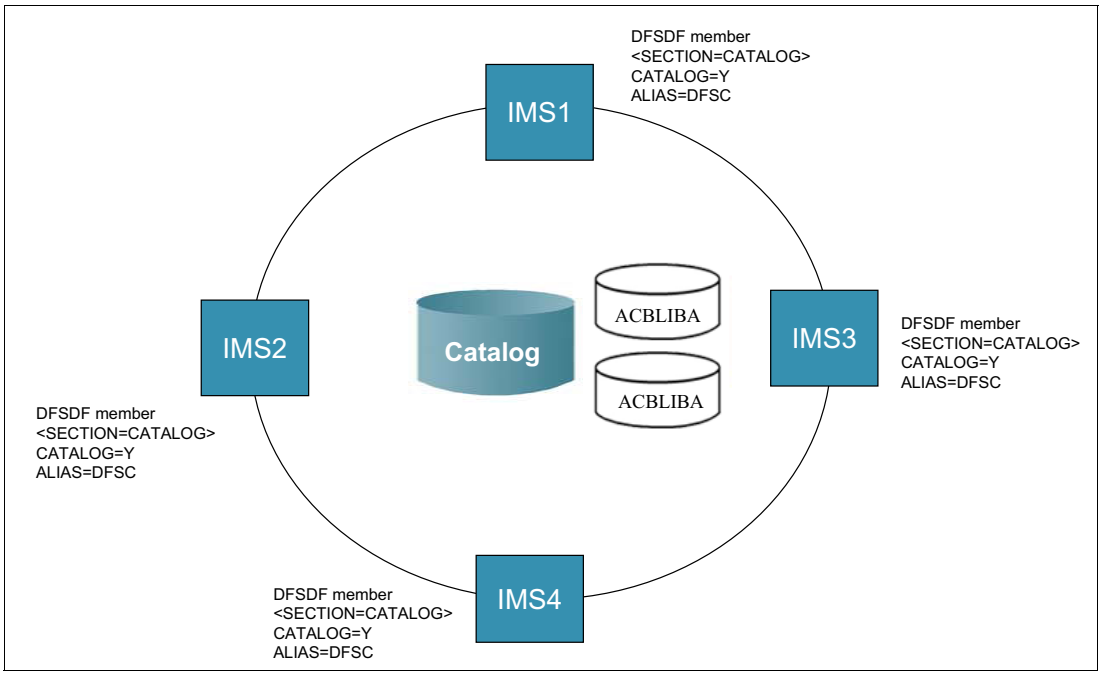


Figure 7 Multiple IMS, shared ACBLIBs, and shared IMS catalog

IMS handles loading the DBDs: DFSCD000 and DFSCX000, and the PSBs: DFSCP000, DFSCP001, DFSCP002, and DFSCP003. IMS also makes the internal changes needed for the aliased names.

Definitions needed for the IMS catalog

Before you can have the IMS catalog created and initially loaded with the populate utility, you need to update the IMS system to use it. The definition is done by adding statements to the IMS.PROCLIB(DFSDFxxx) member.

You have some flexibility in how to define the CATALOG section in the DFSDFxxx member. A simple single IMS is shown in Example 13 with its suggested ALIAS name.

Example 13 Single IMS system

```
<SECTION=CATALOG>
CATALOG=Y
ALIAS=DFSC
```

For example, if you have two or more IMS systems that use shared queues, you might want them to share a single IMS catalog and the same DFSDFxxx proclib member. In this case, you can define different section suffixes in the CATALOG statement by using the IMS ID. And you can define the catalog database with a single shared ALIAS name. See Example 14.

Example 14 Multiple IMS systems

```
<SECTION=CATALOGI12A>
CATALOG=Y
ALIAS=I12Q
<SECTION=CATALOGI12C>
CATALOG=Y
ALIAS=I12Q
```

Application use of the catalog

IMS 12 can access the IMS catalog when IMS metadata is required. IMS 12 also provides a user interface to the IMS catalog data and a new DL/I call to access the IMS catalog. One source of metadata for populating the catalog is the DBDs and PSBs. The extensions of the metadata information and the additional data manipulation possibilities that are introduced, require additional information from the DBD and PSB sources. As a result, several DBDGEN and PSBGEN macros are enhanced with IMS 12.

We provide details about using the IMS catalog in the following sections:

- DBD and PSB source changes
- Get Unique Record DL/I call
- IMS catalog access
- SSA enhancements

DBD and PSB source changes

There are new DBDGEN statements in IMS 12 and changes to existing ones. The new DBDGEN statements are also represented in the metadata information in the catalog. Much of this metadata information is the basis for the extended Object exploitation by the Java language.

DFSMARSH statement in the DBD

The DFSMARSH statement in a DBD defines the marshalling attributes for field data. The DFSMARSH statement must immediately follow the FIELD statement to which it applies. You can use the DFSMARSH statement to specify the following data marshalling attributes for the data contained in a field:

- You can specify the code page or character encoding that defines the character data in a field.
- You can specify a data-conversion routine for IMS to use when converting field data. For example, use it when converting field data from the data type that IMS uses to physically store data, to a data type expected by an application program.
- You can specify whether a decimal data type is signed or not with the ISSIGNED parameter.
- The pattern to use for dates and times can be specified with the PATTERN parameter.
- You can specify the properties that are used with a user-provided data-conversion routine.

The MAR segment type in the catalog database contains information about a field marshaller definition in an IMS database. Each FLD segment can have a MAR child segment that contains data marshalling properties for that field. The following information in this segment type is generated from the input parameters of the DFSMARSH statement of the DBDGEN utility:

- ENCODING

  Specifies the default encoding of all character data in the database defined by this DBD.

  Default ENCODING is CP1047 (EBCDIC). This default value can be overridden for individual segments and fields.

- USERTYPECONVERTER

  Specifies the fully-qualified Java class name of the user-provided Java class to be used for type conversion.

- INTERNALTYPECONVERTER

  Specifies the internal conversion routine that IMS uses to convert the IMS data into Java objects for Java application programs. IMS requires the specification of either INTERNALTYPECONVERTER or USERTYPECONVERTER, but not both. Valid values for the INTERNALTYPECONVERTER parameter include:

  ARRAY, BINARY, BIT, BLOB, BYTE, CHAR, CLOB, DOUBLE, FLOAT, INT, LONG, PACKEDDECIMAL, SHORT, STRUCT, UBYTE, USHORT, UINT, ULONG, XML_CLOB, and ZONEDDECIMAL.

- ISSIGNED

  Valid only for DATATYPE=DECIMAL. Valid values are Y (default) or N.

- PROPERTIES

  Specifies properties for user type converter names with the USERTYPECONVERTER parameter. These properties are passed to the user type converter.

- PATTERN

  An optional field that specifies the pattern to use for the date, time, and time stamp Java data types.

  The PATTERN parameter applies only when the DATE, TIME, or TIMESTAMP is specified on the DATATYPE keyword in the FIELD statement. Also, CHAR must be specified on the INTERNALTYPECONVERTER keyword in the DFSMARSH statement.

DFSMAP statement

The DFSMAP statement enables the alternate mapping of fields within a segment.

The DFSMAP statement defines a group of one or more map cases and relates the cases to a control field. The control field identifies which map case is used in a particular segment instance, as shown in Figure 8.

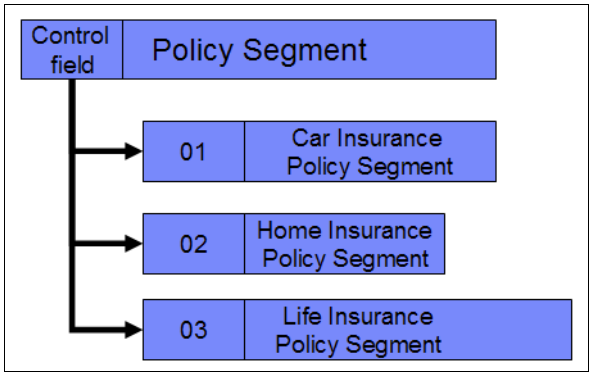


Figure 8 Several mappings for the same segment

- NAME

  Defines the name of this map.

- DEPENDING ON

  External name of the control field within this segment that contains the value that determines which map case is used for a particular segment instance. If the control field does not contain a value that corresponds to a CASEID in a DFSCASE statement for this map, the map is not used for this segment instance.

  If the FIELD statement that defines the control field does not explicitly code the EXTERNALNAME parameter, specify the value of the NAME parameter in the DEPENDING ON field.

- REMARKS

  Allow for a comment up to 256 characters long to be recorded in the DBD and the IMS catalog database.

DFSCASE statement

Many applications allow for several different layouts for the same segment type in an IMS database. Typically, this configuration is done in COBOL with the REDEFINES statement, and in PL/I with the DEFINED statement. The corresponding facility in IMS 12 is the DFSCASE statement, which defines a map case: A conditional mapping of a set of fields in a segment. Each map case has a name and an identifier (ID). A single segment can be defined with multiple map cases. The map cases within a segment are grouped by the DFSMAP statement, which also associates the map cases with their control field.

Each map case is defined with a case ID. The case ID is stored in the control field to identify which case map is used for a particular segment instance. Typically, the control field is defined at the beginning of the segment, before the DFSMAP statement.

The fields that make up a map case identify the map case that they belong to by the name on the CASENAME parameter in the FIELD statement. CASENAME is valid and required only to associate a FIELD statement with the preceding DFSCASE statement that defines the map case to which this field belongs. The value of CASENAME must match the value specified on the NAME parameter of the DFSCASE statement.

The ID of a map case is specified on the CASEID parameter. In a segment instance, the ID is inserted in a control field to indicate which map case is in use:

- NAME

  Defines the name of this case.

- CASEIDTYPE

  Defines the data type of the value specified in the CASEID parameter.

- MAPNAME

  The name of the map that this case belongs to, as specified on the NAME parameter in the DFSMAP statement.

- CASEID

  Defines a unique identifier for the case. A segment instance specifies the CASEID value in a user-defined control field when part or all of the field structure of the segment is mapped by this case. The CASEID must be unique within the map to which the case belongs.

- REMARKS

  Allow for a comment of up 256 characters to be recorded in the DBD and the IMS catalog database.

Enhancements to DBD definition statements

There are a number of enhancements to the statements that are used to describe a database definition with IMS 12:

Changes to the DBD statement

- ENCODING

  Specifies the default encoding of all character data in the database that is defined by this DBD otherwise default ENCODING is CP1047 (EBCDIC). The value can be overridden in individual segments or fields.

Changes to the SEGM statement

- ENCODING

  Specifies the default encoding of all character data in the database that is defined by this DBD otherwise default ENCODING is CP1047 (EBCDIC). The value can be overridden in individual fields.

- EXTERNALNAME

  This parameter is used to give a segment an extended (1 - 128 byte) name.

Changes to the FIELD statement

- NAME

  Specifies the name of this field within a segment type. The name must be provided to be able to search on the field.

  For key-sequenced field types and field types that are referenced by an XDFLD statement, the NAME parameter is required.

  For other field types, you can optionally omit the NAME parameter when the EXTERNALNAME parameter is specified.

- PARENT

  Specifies the name of a field that is defined as a structure or array in which this field is contained. Must be the value of the EXTERNALNAME parameter in the definition of the referenced field.

  The referenced field must be defined with either DATATYPE=ARRAY or DATATYPE=STRUCT.

- CASENAME

  The name of the map case to which this field belongs.

  This parameter is required only to associate a field statement with the preceding DFSCASE statement that defines the map case.

  This parameter must match the value specified on the NAME parameter of the DFSCASE statement.

- DATATYPE

  This parameter is an optional value that specifies one of the following external data types for the field:

  ARRAY, BINARY, BIT, BYTE, CHAR, DATE, DECIMAL (with precision and scale), DOUBLE, FLOAT, INT, LONG, OTHER, SHORT, STRUCT, TIME, TIMESTAMP, and XML.

- REDEFINES

  This is a field that specifies the name of another field in the segment. The field must be the same length as the field that is redefined.

  The field cannot be an ARRAY or contain an ARRAY.

- EXTERNALNAME

  This parameter is an optional alias for the NAME= parameter. Java application programs use the external name to refer to the field.

  The EXTERNALNAME parameter is required only when the NAME parameter is not specified. If the NAME parameter is not specified, you cannot search on this field.

- DEPENDSON

  This parameter specifies the name of a field that defines the number of elements in a dynamic array. Referenced fields must precede the field statement that specifies this parameter.

  The name specified must be the value, whether explicitly defined or accepted by default, of the EXTERNALNAME parameter in the definition of the referenced field.

  The DEPENDSON parameter is valid only when DATATYPE=ARRAY is also specified.

  The field referenced by the DEPENDSON parameter must be defined with one of the following DATATYPE values: INT, SHORT, LONG, DECIMAL.

- MINOCCURS

  Valid for the DATATYPE=ARRAY only, a MINOCCURS is a required numeric value that specifies the minimum number of elements in an ARRAY. MINOCCURS is invalid for all other data types.

- MAXOCCURS

  Valid for the DATATYPE=ARRAY only, a MAXOCCURS is a required numeric value that specifies the maximum number of elements in an ARRAY.

  MAXOCCURS must be greater than or equal to MINOCCURS, but not zero.

- MAXBYTES

  This parameter specifies the maximum size of a field in bytes when the byte-length of the field instance can vary based on the number of elements in a dynamic array. MAXBYTES and BYTES are mutually exclusive.

  The value of MAXBYTES must be greater than or equal to the maximum total of the byte values of all fields nested under this field.

  The MAXBYTES parameter is required and valid only in the following cases:

  – The field is defined as a dynamic array.
  – For a field defined as a static array or for a structure that contains a nested field that is defined as a dynamic array.

- STARTAFTER

  This parameter specifies the name of the field that directly precedes this field in the segment. The name specified must be the value of the EXTERNALNAME parameter in the definition of the referenced field.

  STARTAFTER is required and valid only when the starting position of a field cannot be calculated. The starting position cannot be calculated because the field is preceded at a prior offset by a field defined as a dynamic array.

  The STARTAFTER parameter cannot be specified on fields that define an array field as a parent.

- RELSTART

  This parameter specifies the starting position of a field that is defined as an element of an array or a structure:

  – For fields that specify an array field as a parent, RELSTART is required.
  – For fields that specify a structure as a parent, RELSTART is required only when a dynamic array precedes the structure at any prior offset in the segment.

  RELSTART is the starting byte offset of the field relative to the start of the array or structure.

Changes to remarks for DBDs and PSBs

The following existing IMS macros are updated to allow up to 256 characters to be stored as a remark for a DBD or PSB. This update allows the user to specify comments that are stored in the DBDLIB, PSBLIB, ACBLIB, and IMS catalog. This notation helps to ensure that comments are not lost if the source is lost:

- PSB remarks

  PCB, SENFLD, SENSEG

- DBD remarks

  DBD, SEGM, FIELD, XDFLD, LCHILD, DATASET, AREA

Get Unique Record DL/I call

IMS 12 provides a new DL/I call specifically for the IMS catalog. The *Get Unique Record* (GUR) call retrieves the metadata for an IMS database (DB) or PSB from the catalog database. To read all the segments in this database record, the GUR call functions similarly to a Get Unique (GU) call. The GU call is followed by a series of Get Next within Parent (GNP) calls. This call can only be used to retrieve the metadata definition of an IMS DB or PSB from the catalog database.

The GUR call reads an entire database record from the catalog database and returns an XML document that contains the metadata definition for the requested IMS resource (DBD or PSB). If the XML document is larger than the IOAREA provided by the application, subsequent GUR calls can be issued to retrieve the balance of the XML document. This process is done by passing back a token that was returned by the initial GUR call. The data is buffered by IMS. This buffering is based on the assumption that the entire XML document is needed to minimize the processor usage when you ask for subsequent blocks of data to be returned. The advantage is that you do not need to over-estimate the size of the IOAREA. If the IOAREA is too small, the call still succeeds and the balance of the data can be requested without having to reread the catalog database. This new call is designed to simplify access to the database definition metadata and minimize the processor usage when retrieving it.

If you pass a token with a value of zero, IMS assumes that it is a new GUR call and starts from the beginning.

DFSDDLT0 and IMS Restructured Extended Executor (REXX) are updated to add special processing for the IMS catalog and the XML that is returned from a GUR call.

The GUR call builds an entire XML instance document from the information in the catalog database. Internally, each GUR call with a zero AIBRTKN executes a GU and multiple GNP calls to read the IMS catalog. When the I/O area is too small for output, the output buffer is kept. A token is returned in the application interface block (AIB) to allow the application to resume copying data from the buffer to I/O area on subsequent calls.

GUR reads an entire record. SSAs can be coded starting with the HEADER and then the DBD or PSB. GUR does not function like GU calls in which you can read a segment with one qualified SSA, then get a lower level segment with a different SSA.

The GUR call can only be used in the IMS catalog database. The call requires the AIB interface. If an attempt is made to use the GUR call through the ASMTDLI, CBLTDLI, or PLITDLI interfaces, IMS returns a “DE” status code.

During initialization, IMS reads the time stamps from the ACBLIB and stores them into the database control blocks. The GUR call uses the time stamp to find the currently active member that is used by IMS. Or, the call finds the first record only when IMS does not have a time stamp. IMS does not have a time stamp for resources that are not defined to this IMS.

Segment search arguments used with the GUR call

Typically, segment search arguments (SSAs) are needed when using the GUR call, to identify the database or PSB definition to be retrieved. SSAs are specified on the GUR call in the same way as other GU calls to IMS, that is, **`field <relational operator> field-value`**.

Example 15 on page 27 shows an application program that is using the GUR call to retrieve the metadata for the S2U1DBD database.

Example 15 Sample application that uses the GUR DL/I call

```
Identification division.
         program-id. gur.
       Environment division.
       Data division.
        Working-storage section.
        01 dli-insert pic x(4) value 'ISRT'.
        01 dli-gur    pic x(4) value 'GUR'.
        01 dli-gu     pic x(4) value 'GU'.
        01 header-ssa.
           02 filler          pic x(8)  value 'HEADER  '.
           02 filler          pic x(1)  value '('.
           02 ssa-field-name  pic x(8)  value 'RHDRSEQ '.
           02 ssa-boolean     pic x(2)  value '= '.
           02 ssa-field-value pic x(16) value 'DBD     S2U1DBD '.
           02 filler          pic x(1)  value ')'.
         01  AIB.
           02 AIBRID             PIC x(8).
           02 AIBRLEN            PIC 9(9) USAGE BINARY.
           02 AIBRSFUNC          PIC x(8).
           02 AIBRSNM1           PIC x(8).
           02 AIBRSNM2           PIC x(8).
           02 AIBRESV1           PIC x(8).
           02 AIBOALEN           PIC 9(9) USAGE BINARY.
           02 AIBOAUSE           PIC 9(9) USAGE BINARY.
           02 AIBRESV2           PIC x(12).
           02 AIBRETRN           PIC 9(9) USAGE BINARY.
           02 AIBREASN           PIC 9(9) USAGE BINARY.
           02 AIBERRXT           PIC 9(9) USAGE BINARY.
           02 AIBRESA1           USAGE POINTER.
           02 AIBRESA2           USAGE POINTER.
           02 AIBRESA3           USAGE POINTER.
           02 AIBRESV4           PIC x(40).
           02 AIBRSAVE     OCCURS 18 TIMES USAGE POINTER.
           02 AIBRTOKN     OCCURS 6 TIMES  USAGE POINTER.
           02 AIBRTOKC           PIC x(16).
           02 AIBRTOKV           PIC x(16).
           02 AIBRTOKA     OCCURS 2 TIMES PIC 9(9) USAGE BINARY.
        01 gur-returned-data pic x(32000).
        Linkage section.
         1 io-pcb.
           3 filler          pic x(60).
         1 db-pcb.
           3 filler          pic x(10).
           3 status-code     pic x(2).
           3 filler          pic x(20).
       Procedure division using io-pcb db-pcb.
           move "DFSAIB  "    to AIBRID.
           move length of aib to AIBRLEN.
           move length of gur-returned-data to AIBOALEN.
           move spaces        to AIBRSFUNC.
           MOVE "DFSCAT00"    TO AIBRSNM1.
           call 'AIBTDLI' using dli-gur aib
                 gur-returned-data header-ssa.
```

```
           set address of db-pcb to aibresa1.
           display '|' status-code '|' db-pcb.
           display 'aib return is ' AIBRETRN .
           display 'aib reason is ' AIBREASN .
           display gur-returned-data.
           goback .
       End program gur.
```

The first few lines of the data returned by the GUR call from the program (shown in Example 15 on page 27), are shown in Example 16. The first 56 characters in the IOAREA are not part of the XML document.

Example 16 Sample data returned by a GUR call

```
_%      ?>       > ?  >            /> /%?>   `     <ns2:dbd xmlns:ns2="http://www.ibm.com/ims/DBD" dbdName="S2U1DBD
" timestamp="1215015125765" version="02/10/1114.51" xmlSchemaVersion="1"><access dbType="HIDAM"><hidam datxexit="N" pass
word="N" osAccess="VSAM"><dataSetContainer><dataSet  ddname="S2U1DB" label="DSG1" searchA="0" scan="3"><block size="0"/>
<size size="0"/><frspc fspf="0" fbff="0"/></dataSet></dataSetContainer></hidam></access><segment imsName="CUSTROOT" name
="CUSTROOT" encoding="Cp1047"><hidam label="DSG1"><bytes maxBytes="76"/><rules insertionRule="L" deletionRule="L" replac
ementRule="L" insertionLocation="LAST"/><pointer physicalPointer="TWINBWD" lparnt="N" ctr="N" paired="N"/></hidam><field
 imsDatatype="C" imsName="CUSTNO" name="CUSTOMERNUMBER" seqType="U"><startPos>1</startPos><bytes>4</bytes><marshaller><t
ypeConverter>BINARY</typeConverter></marshaller><applicationDatatype datatype="BINARY"/></field><field imsDatatype="C" n
ame="FIRSTNAME"><startPos>5</startPos><bytes>10</bytes><marshaller encoding="Cp1047"><typeConverter>CHAR</typeConverter>
</marshaller><applicationDatatype datatype="CHAR"/></field><field imsDatatype="C" name="LASTNAME"><startPos>15</startPos
><bytes>20</bytes><marshaller encoding="Cp1047"><typeConverter>CHAR</typeConverter></marshaller><applicationDatatype dat
atype="CHAR"/></field><field imsDatatype="C" name="DATEOFBIRTH"><startPos>35</startPos><bytes>10</bytes><marshaller enco
ding="Cp1047"><typeConverter>CHAR</typeConverter></marshaller><applicationDatatype datatype="CHAR"/></field><field imsDa
tatype="C" name="HOUSENAME"><startPos>45</startPos><bytes>20</bytes><marshaller encoding="Cp1047"><typeConverter>CHAR</t
ypeConverter></marshaller><applicationDatatype datatype="CHAR"/></field><field imsDatatype="C" name="HOUSENUMBER"><start
```

A better way to display this XML information is through a browser. Example 17 shows the same information as displayed by Microsoft Internet Explorer.

Example 17 Sample data returned by a browser

```
<ns2:dbd xmlns:ns2="http://www.ibm.com/ims/DBD" dbdName="S2U1DBD" timestamp="1215015125765"
version="02/10/1114.51" xmlSchemaVersion="1">
<access dbType="HIDAM">
<hidam datxexit="N" password="N" osAccess="VSAM">
<dataSetContainer>
<dataSet ddname="S2U1DB" label="DSG1" searchA="0" scan="3">
<block size="0" />
<size size="0" />
<frspc fspf="0" fbff="0" />
</dataSet>
</dataSetContainer>
</hidam>
</access>
<segment imsName="CUSTROOT" name="CUSTROOT" encoding="Cp1047">
<hidam label="DSG1">
<bytes maxBytes="76" />
<rules insertionRule="L" deletionRule="L" replacementRule="L" insertionLocation="LAST" />
<pointer physicalPointer="TWINBWD" lparnt="N" ctr="N" paired="N" />
</hidam>
```

```
<field imsDatatype="C" imsName="CUSTNO" name="CUSTOMERNUMBER" seqType="U">
<startPos>1</startPos>
<bytes>4</bytes>
<marshaller>
<typeConverter>BINARY</typeConverter>
</marshaller>
<applicationDatatype datatype="BINARY" />
</field>
<field imsDatatype="C" name="FIRSTNAME">
<startPos>5</startPos>
<bytes>10</bytes>
<marshaller encoding="Cp1047">
<typeConverter>CHAR</typeConverter>
</marshaller>
<applicationDatatype datatype="CHAR" />
</field>
```

The XML schemas for the documents returned as responses to this call, are included in the IMS.ADFSSMPL data set:

- DFS3XDBD.xsd (for DBD records)
- DFS3XPSB.xsd (for PSB records)
- SSAs can be specified only up to the DBD or PSB level in the catalog database

  No more than two SSAs can be specified:

  - One for the root HEADER segment
  - One for the DBD or PSB segment

Example 18 shows the IMS Test Program (DFSDDLT0) statements that are needed to process a GUR call. The data statement is needed to provide sufficient IO-AREA for the XML document that contains the database definition to be returned to DSFSDDLT0.

Example 18 DFSDDLT0 statements for GUR

```
S 1 1 1 1 1    DFSCD000      AIB
L        GUR   HEADER  (RHDRSEQ ==DBD     S2U1DBD )
L  Z9999 DATA
```

IMS catalog access

When an IMS application program requires access to the metadata in the catalog, a PSB to access the catalog database is automatically attached to the PSB that is loaded for the application. IMS can then use that PSB to access the metadata in the IMS catalog.

Direct catalog interface

An application user can issue a DL/I call by using the PCB that directly references the IMS catalog DBD if they need to access the metadata.

The application operates as a normal DL/I application. It can, for example, use the new GUR DL/I call to read the whole root and all of the child segments from the IMS catalog.

Indirect catalog interface

The indirect catalog interface is used when a process needs access to the metadata, but is not accessing the IMS catalog directly. The application processes as usual and can issue DL/I calls with a PSB that does not reference with the catalog DBD. When a process needs access to the metadata (in the past, the Java class from the DLIMODEL utility was used), IMS dynamically attaches a PSB with a catalog PCB. Figure 9 shows this access.

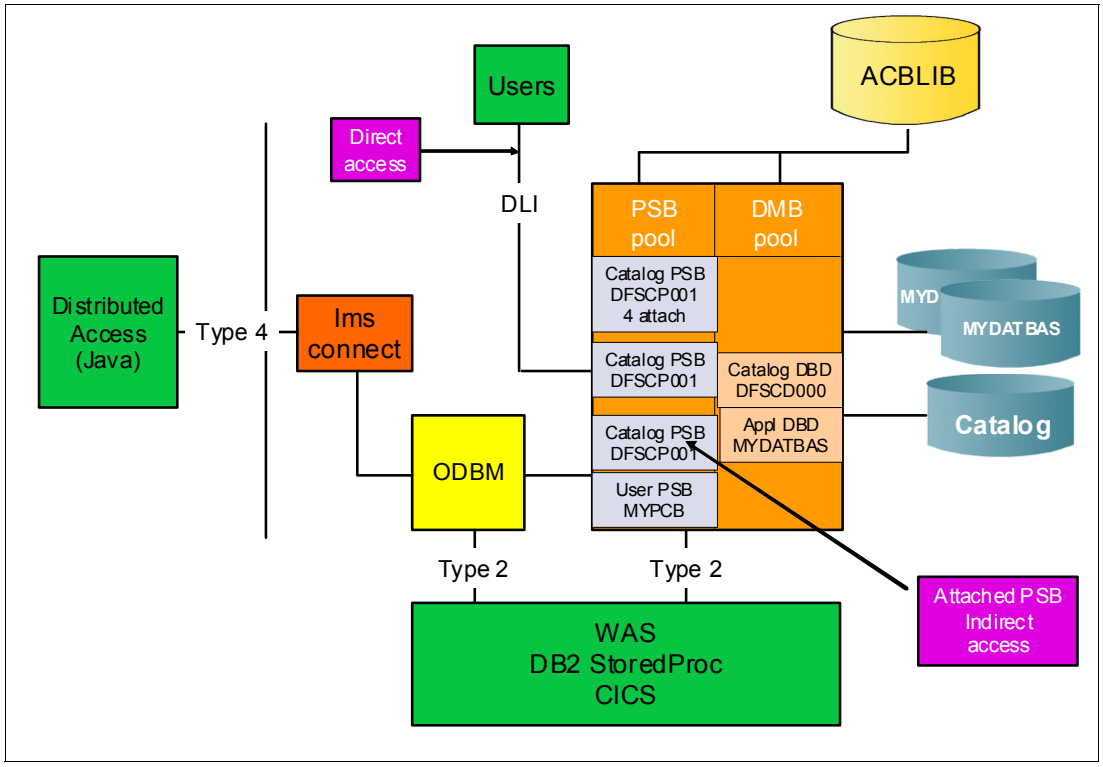


Figure 9 IMS catalog access

The catalog PCB is used by the DL/I engine without any external effects on the calling application. The normal mode of operation is that you are only reading the IMS catalog with a PROCOPT=G (read with integrity). The main catalog PCB is DFSCAT00.

For the message processing program (MPP) applications (running in an online IMS), the dynamic attach of a PSB that contains the IMS catalog PCB, occurs at the first reference during the first DL/I call. The database is accessed with deferred scheduling that only occurs when the first call is made that needs access to the IMS catalog. The IMS catalog database availability does not affect application availability. If the IMS catalog is offline, you can get an abend (U3303), but that does not cause the application to terminate.

In a DL/I batch environment, the dynamic attach of the IMS catalog PCB occurs during the batch initialization. Scheduling is only done once for the batch when the region starts. A batch job loads the IMS catalog PSB for later reference if the catalog is enabled in the DFSDFxxx PROCLIB member.

SSA enhancements

The new SSA format, *get by offset,* allows a new search function by offset and length, instead of field name. Fields are no longer required to be defined in the DBD:

- Support is added for DFSDDLT0 and IMS REXX.
- Performance is the same as a non-key field search.
- IMS scans the database to look for matches.
- SSA contains offset and length followed by operator and value.

In Figure 10, an SSA search on the field type is issued, although the type is not known by the DBD.

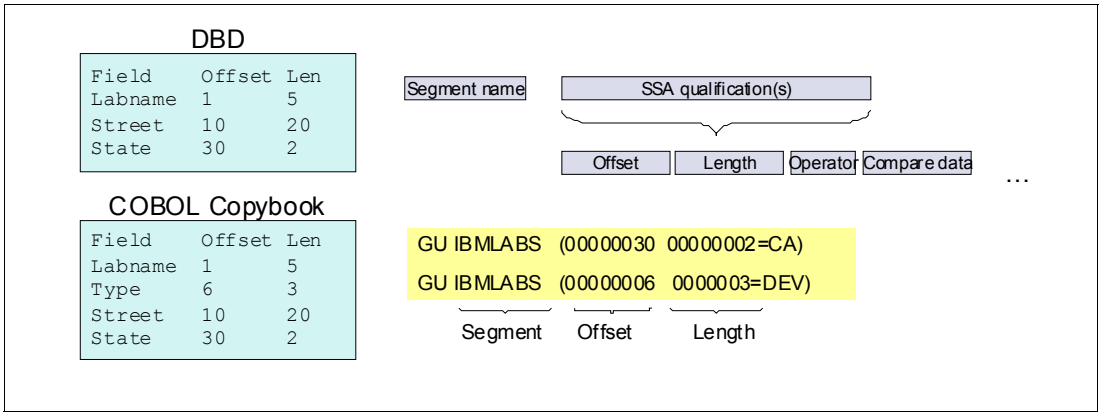


Figure 10 Using get by offset

Field level sensitivity allows the user to change the layout of the segment that is returned in the I/O area. The fields can be reordered, omitted, or spaces can be added between, as shown in Figure 11. The new SSA qualifications can be used in combination with the existing SSA qualification format. For example, the following SSA combination is valid for the fields: labname and type.

```
GU IBMLABS*O (LABNAME =SVL  &00000006 00000003=DEV)
```

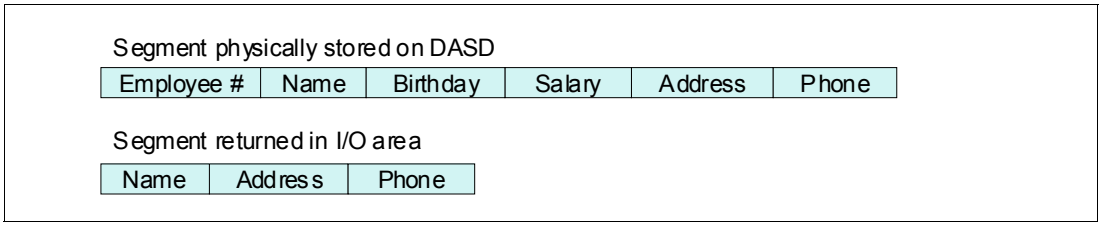


Figure 11 Field sensitivity

In this request, the SSA offset is relative to the physical starting position in the segment that is visible to your program. If you attempt to search for fields that are outside of the sensitive area (that is, past the end of the segment), IMS returns an AK status code. If a segment is not found that matches the SSA qualifications, a GE or GB status code is returned.

The role of the IMS Enterprise Suite Explorer for Development

The IMS Enterprise Suite Explorer (*IMS Explorer*) for Development is an Eclipse-based graphical tool that simplifies IMS application development tasks. The tool simplifies tasks such as updating IMS database and program definitions, and by using standard Structured Query Language (SQL) to manipulate the IMS data.

You can use the IMS Explorer graphical editors to import, visualize, and edit IMS database and program definitions. You can also use the IMS Explorer to easily access and manipulate data that is stored in IMS, by using standard SQL.

The IMS Explorer can also directly import DBDs and PSBs, or can obtain existing catalog information from the IMS catalog via a type 4 connection.

The population of the IMS catalog is mainly done from the PSB and DBD sources. Additional DBDGEN statements and additional attributes on the FIELD macro can provide more complete metadata information.

Additional metadata can be collected from COBOL copybooks and PL/I or C include members. After updating the metadata information in the PSBs and DBDs in the Explorer, those sources can be sent to the host for the GEN process. This process causes the metadata to be populated into the IMS catalog.

Figure 12 shows how the Explorer is able to interact with the IMS catalog directly, via a type 4 driver connection. Figure 12 also shows an interaction with the catalog indirectly, via a File Transfer Protocol (FTP) import and export.

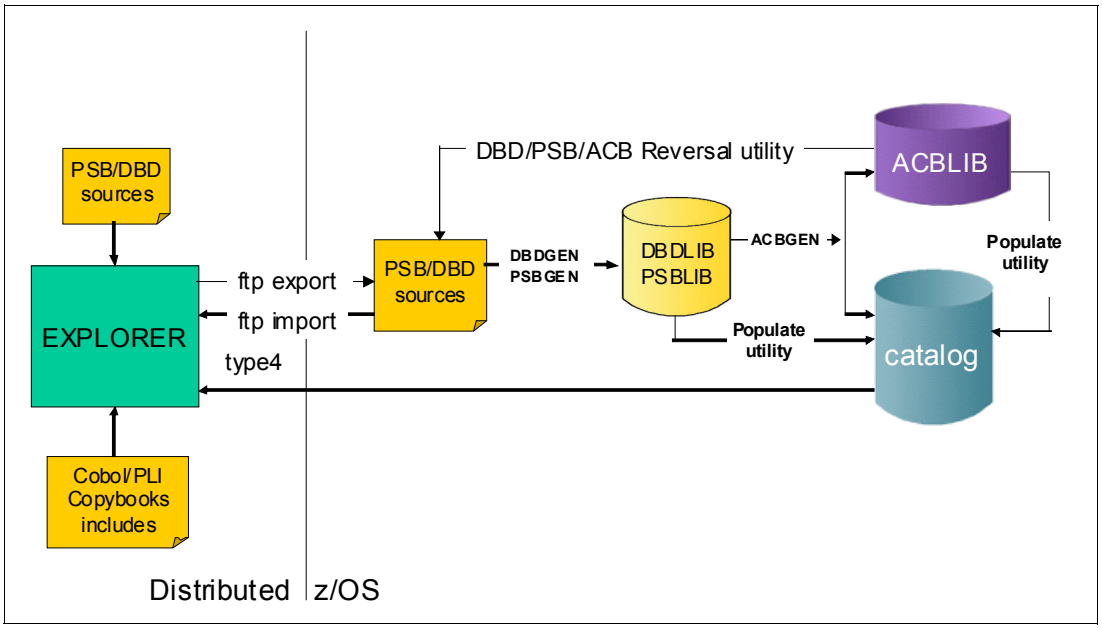


Figure 12 IMS Explorer and catalog

A direct update from the explorer is not available. An intermediate application control block generation (ACBGEN) with population of the catalog is required.

The DLIModel utility is a helpful tool for visualizing. The tool documents all details about the DBDs and PSBs sources. The DLIModel utility is required to produce the `com.ibm.ims.db.DLIDatabaseView` class for accessing the IMS databases from Java applications on the remote and local locations. This access is through the type 4 and type 2 drivers. Both the traditional DL/I SSA and JDBC access can be used.

The DLIModel utility is not changed in Enterprise Suite V 2.1, but its functions are integrated into the IMS Enterprise Suite Explorer. The IMS Explorer uses the centralized IMS catalog on IBM z/OS®. The metadata that is currently available in the DatabaseView class is integrated in the IMS catalog; many more details are kept. The concept of the IMS catalog makes the metadata more dynamic and shared. Its implementation avoids the need for distributing the class to all sites where it might be used in Java programs with DL/I (also via JDBC) access. The access is through the IMS database type 2 and type 4 Universal drivers. The drivers are adapted to integrate this dynamism.

The following features summarize the IBM Enterprise Suite Explorer:

- Incorporates DLIModel functionality

  Support for migration of DLIModel projects is provided
- Provides the following graphical editors for development and visualization:
  - Program Specification Block (PSB)
  - DataBase Description (DBD)
- Shows a relational view of IMS data with graphical assistance to build SQL statements

The IMS Universal JDBC type 4 connectivity extracts PSB and DBD information for local enhancement.

Input for the IMS Explorer

The section, “The role of the IMS Enterprise Suite Explorer for Development” on page 32, explains how the input for the IMS Explorer is taken from local PSB and DBD sources.

The IMS Explorer can also take input from other sources:

- Import of the DLIModel utility project
- Remote import
- From the IMS catalog

  The IMS catalog is the final location where all information about the PSBs and DBDs is consolidated. The Explorer, by using the new type 4 driver, can extract from the catalog PSB and DBD information that is updated locally. After the update, this information is sent back to the consolidating host by FTP. Figure 13 on page 34 shows the Export panel.

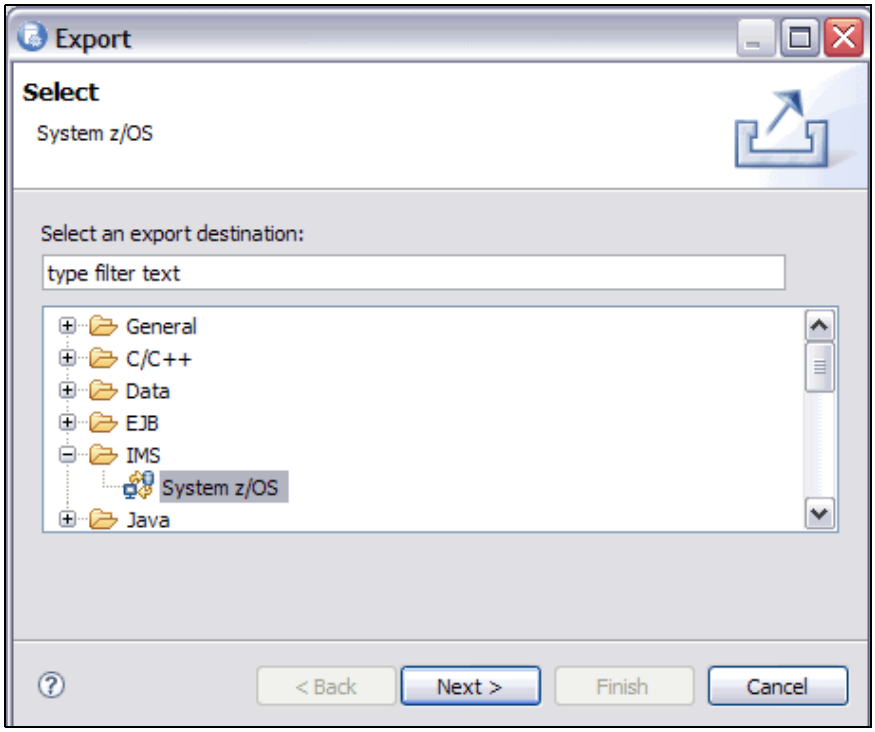


Figure 13 Export to the z/OS host with FTP

A mandatory ACBGEN of the changed PSB or DBD updates the ACBLIB and also populates the updated information to the catalog.

IMS Enterprise Suite Explorer and the IMS catalog

The IMS catalog is a PHIDAM database that contains trusted metadata for IMS databases and applications. All the information that is contained in the runtime ACBLIB (also DBDLIB and PSBLIB) is available to users in the IMS catalog.

The explorer is an important tool for using and updating the information in the catalog. The Enterprise Suite Explorer can pull information from the catalog database, work with it, update it, and finally send it back to the z/OS for consolidation in the catalog.

The following data connections can be used between the Explorer and catalog information:

- From z/OS:
  - FTP import from the DBD and PSB sources
  - Via type-4 driver direct access to the catalog
- To z/OS: FTP export to the DBD and PSB sources

Extending IMS database definitions with the IMS Explorer

This section describes the extension of an IMS DBD with the IMS Explorer for Development. Using the IMS Explorer, you can quickly and easily add metadata information to the IMS databases. You can do this extension by using the COBOL copybooks or PL/I include members that are used by application programs.

Using IMS Explorer to capture IMS metadata

The IMS Explorer facilitates the capture of more metadata for an IMS database than previously was stored in the IMS DBD. With Explorer, the layout of the data in the database segments can be captured and added to the DBD. The enhanced DBD can then be generated back on the host system and included in the catalog. This section details how to use the IMS Explorer for Development to capture the extra metadata information from the COBOL copybooks. This section also describes how the enhanced DBD is ready for generation on the host system.

Before starting, you need to install the IMS Explorer for Development. The code is available as part of the IMS Enterprise Suite Version 2, freely downloadable from the IMS home page (http://www.ibm.com/ims). There are several runtime versions of this code. The code needed for this task is the *shell-sharing* version. This version installs with the IBM Installation Manager as an extension to IBM Rational® Developer for zSeries® or IBM Rational Developer for zEnterprise®.

To begin the process, run Rational Developer for zEnterprise (or Rational Developer for zSeries), and open the IMS Explorer perspective, as shown in Figure 14 on page 36.

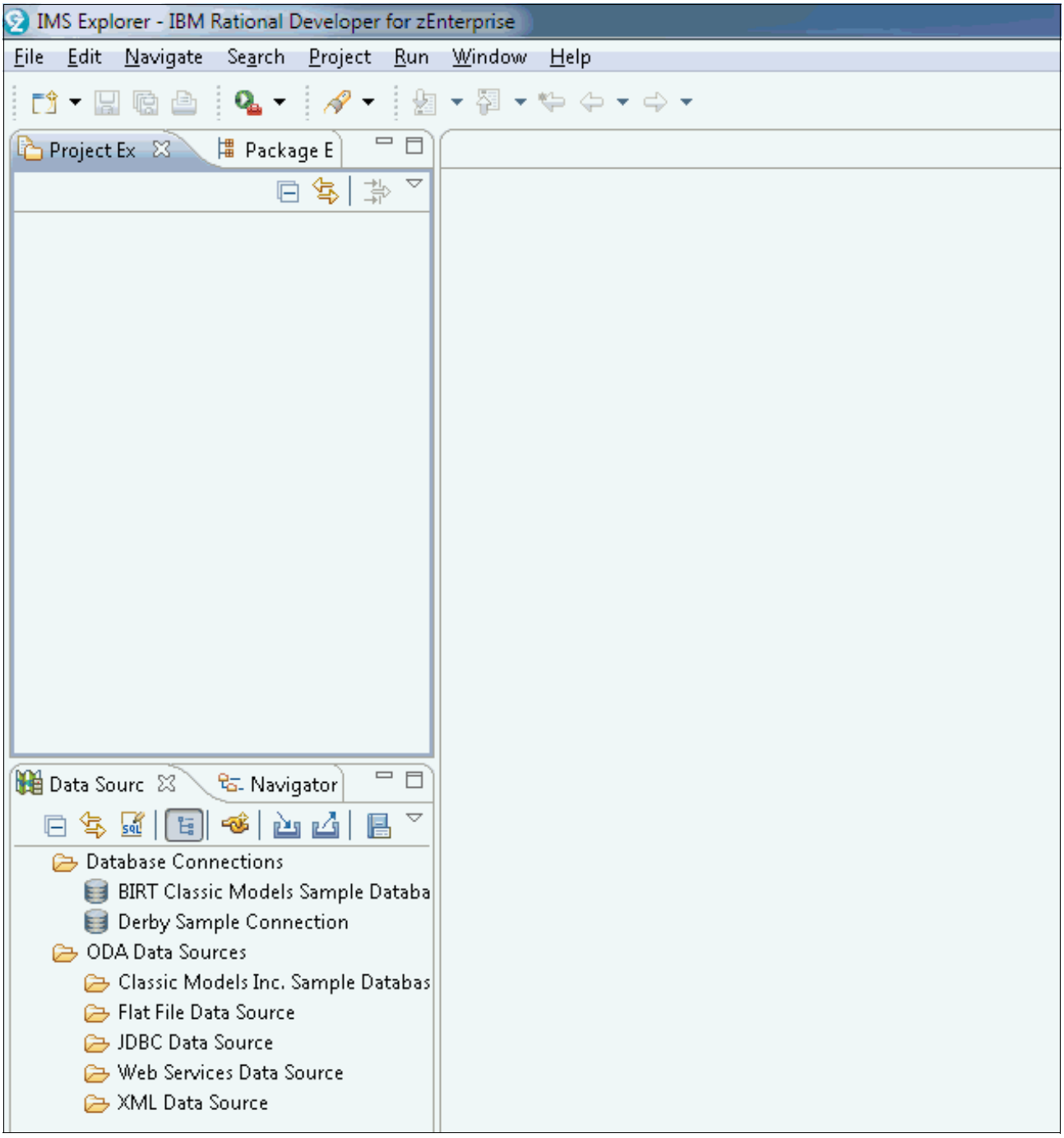


Figure 14 IMS Explorer perspective in Rational Developer for zEnterprise

To create an IMS Explorer Project, right-click in the Project Explorer window (upper left), and select **New** → **Project** to start the Create a New Project wizard (Figure 15 on page 37).

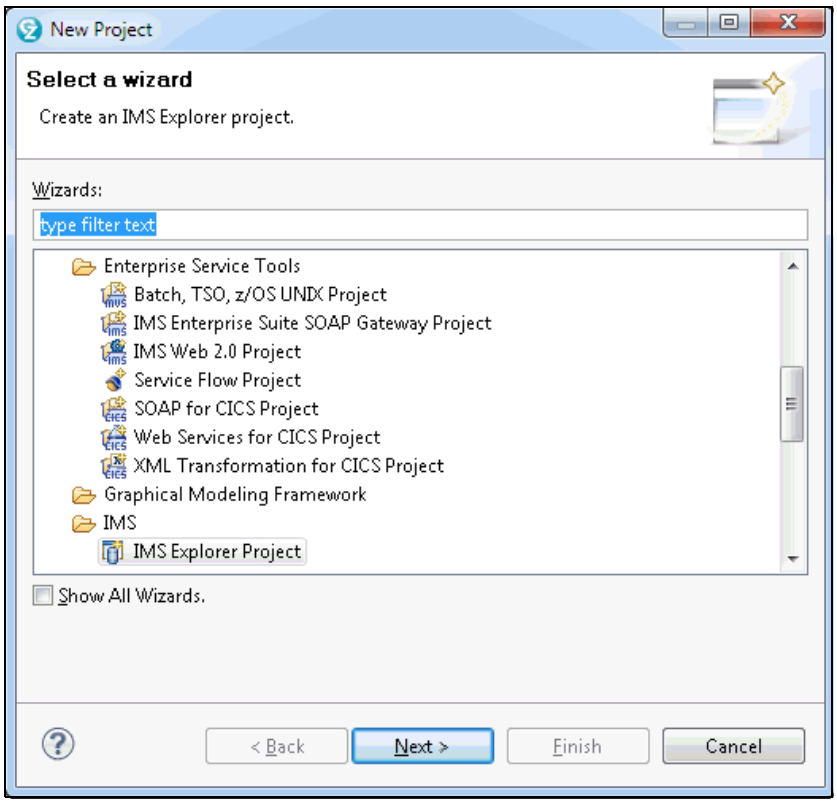


Figure 15 Create a New Project wizard

The wizard is the IMS Explorer Project wizard. Select this one, and click **Next**. See Figure 16.

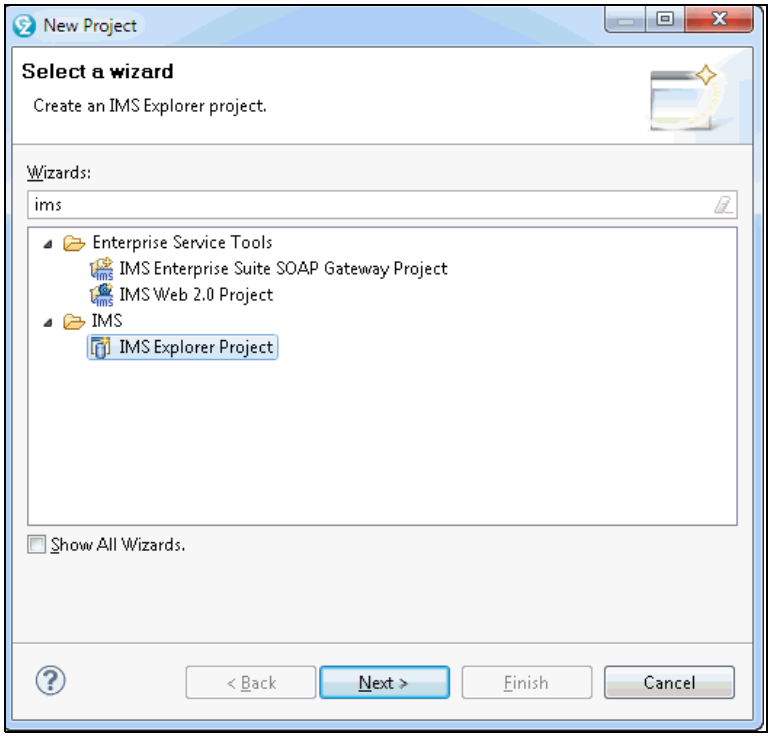


Figure 16 IMS Explorer Project

Enter a name for the IMS Explorer Project in the pane of Figure 17 on page 38.

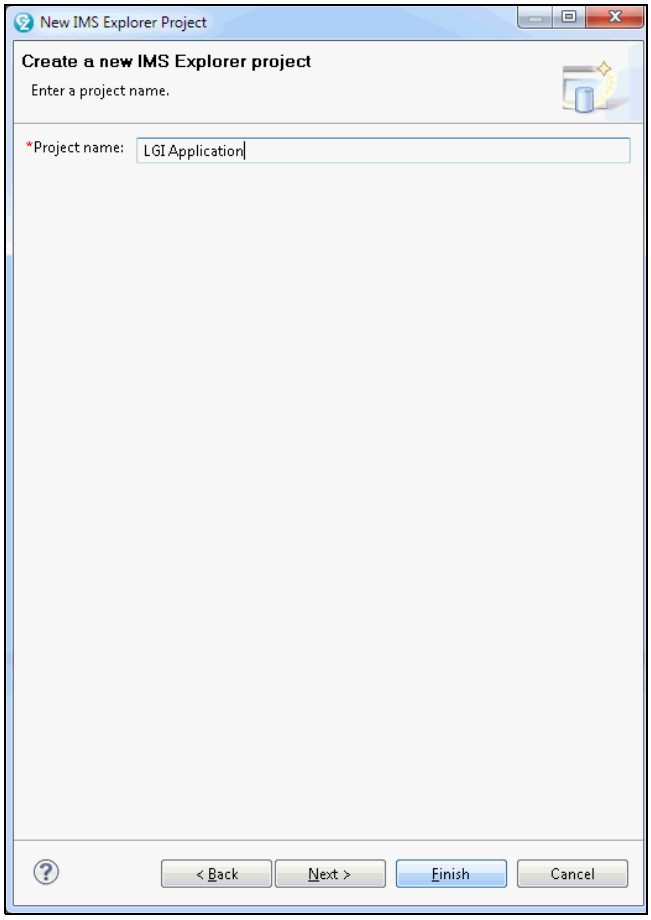


Figure 17 Name the IMS Explorer Project

When the wizard completes, Rational Developer for IBM System z® (RDz) shows the named project in the Project Explorer window (see Figure 18 on page 39).

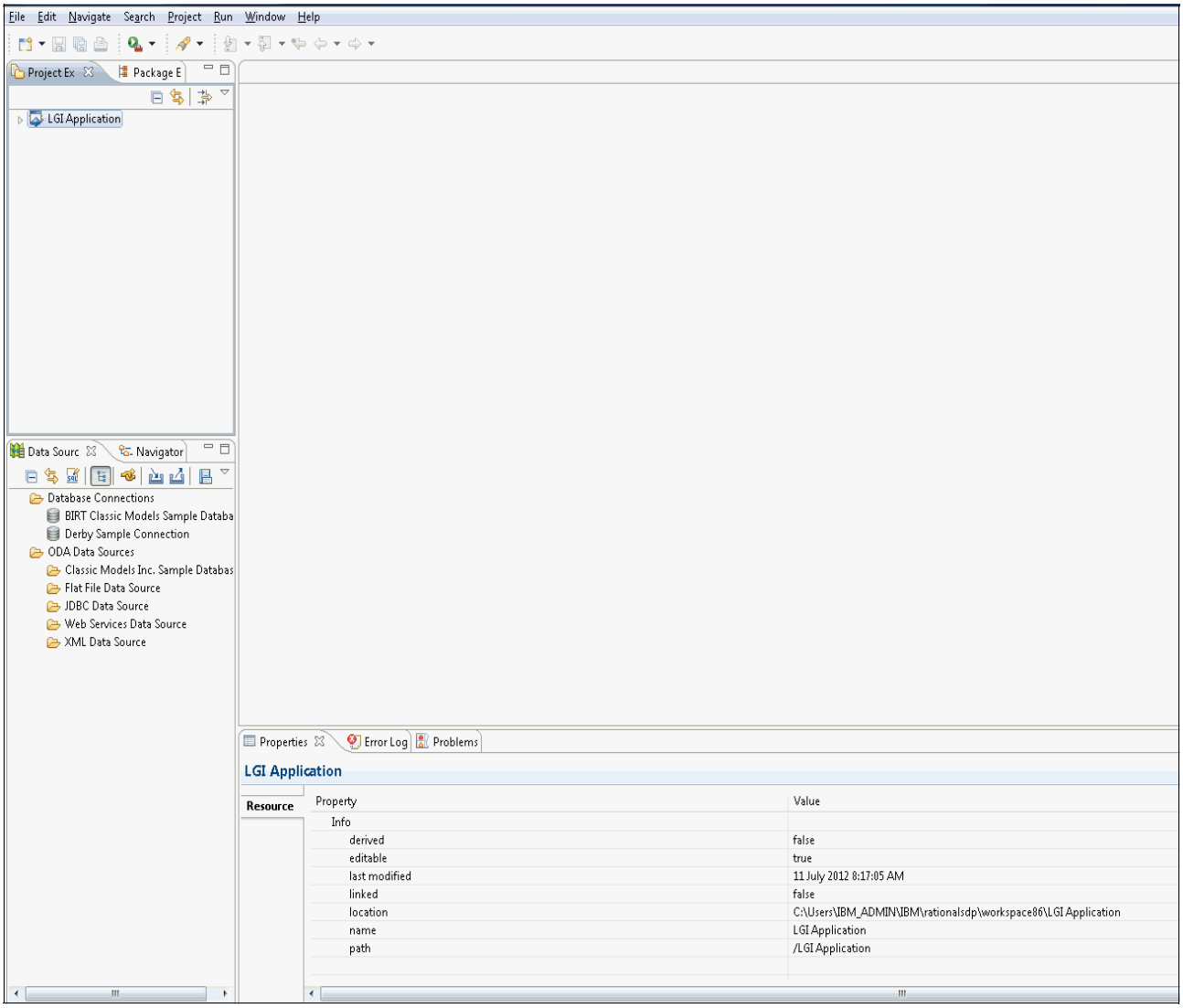


Figure 18 The LGI Application IMS Explorer Project

Next, you must import the DBD for the databases for which you are going to capture the metadata. To import the DBD, right-click the Explorer Project (in this case, the LGI Application), and select **Import** (Figure 19 on page 40).

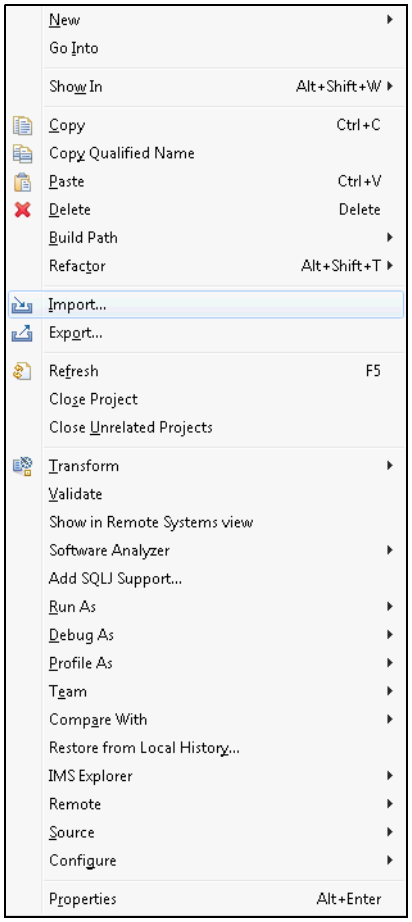


Figure 19 Options for an IMS Explorer Project

Select the **IMS Resources** option, and click **Next**. See Figure 20.

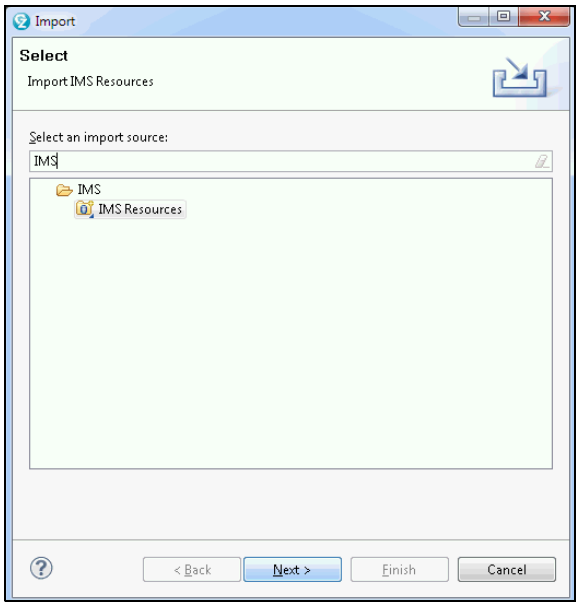


Figure 20 Select an import source

Enter the name of the project in Figure 21 on page 41.

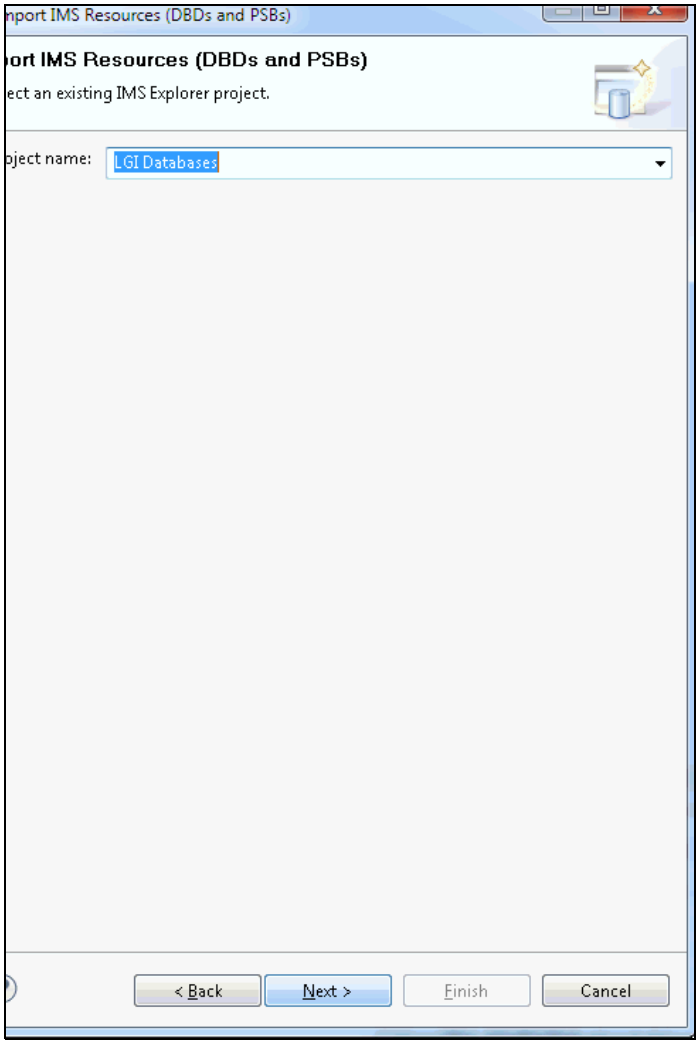


Figure 21 Provide a name for the IMS Resources to be imported

In Figure 22 on page 42, select a location from which the resources are to be imported. The location can be a file on your local workstation, a file on a host in which you are connected, or an IMS catalog. Then click **Next**.

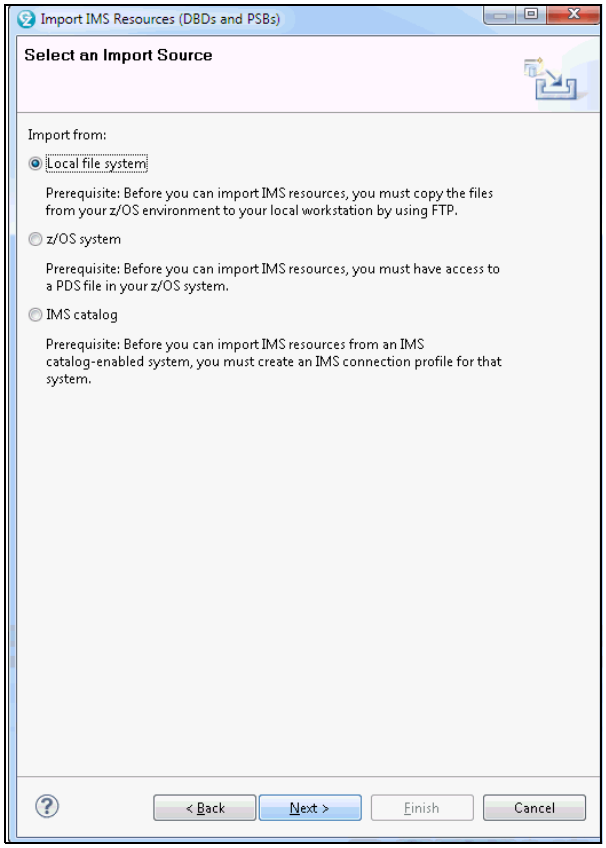


Figure 22 Select the Import Source of the source to be imported

You can now select the files from the source that you want imported into your project. Although there are separate radio buttons on this window for DBDs and PSBs, IMS Explorer can identify the resources in the source file selected. It then lists them in the appropriate list in this window. IMS Explorer also understands a file of JCL containing multiple DBDs, PSBs, or a mixture of both, which simplifies the import process.

The first resources to be imported here are DBDs. Click **Add DBD**, shown in Figure 23 on page 43.

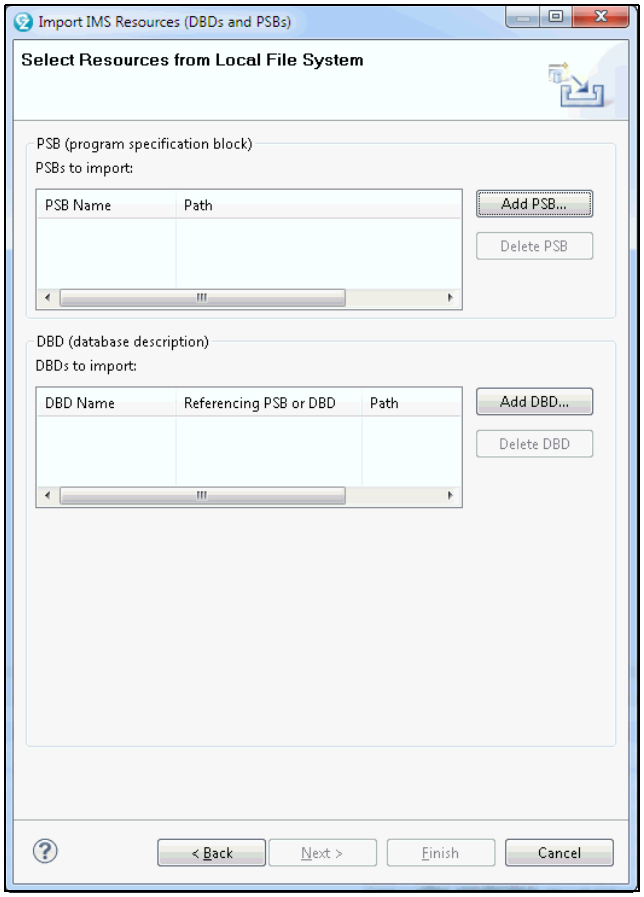


Figure 23 Select resources from the local file system

Select the file that contains the DBDs to be imported. See Figure 24. In this example, multiple DBDs are in a JCL file which was previously downloaded to the local workstation. IMS automatically filters out the JCL and imports each of the DBDs in the file.

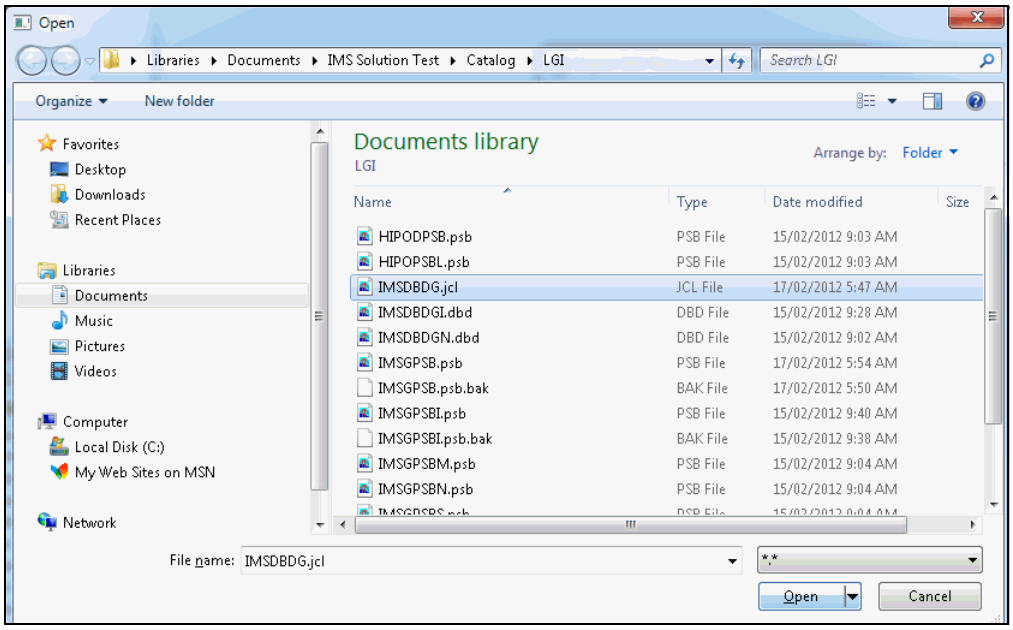


Figure 24 Select the file that contains the resources to be imported

The Import IMS Resources window shows the two DBDs (S2U1DBD and S2U1DXD) which were in the single JCL file that was imported. Any of these DBDs can now be deleted from the import list if wanted.

To add PSBs to this import, click **Add PSB**. See Figure 25.

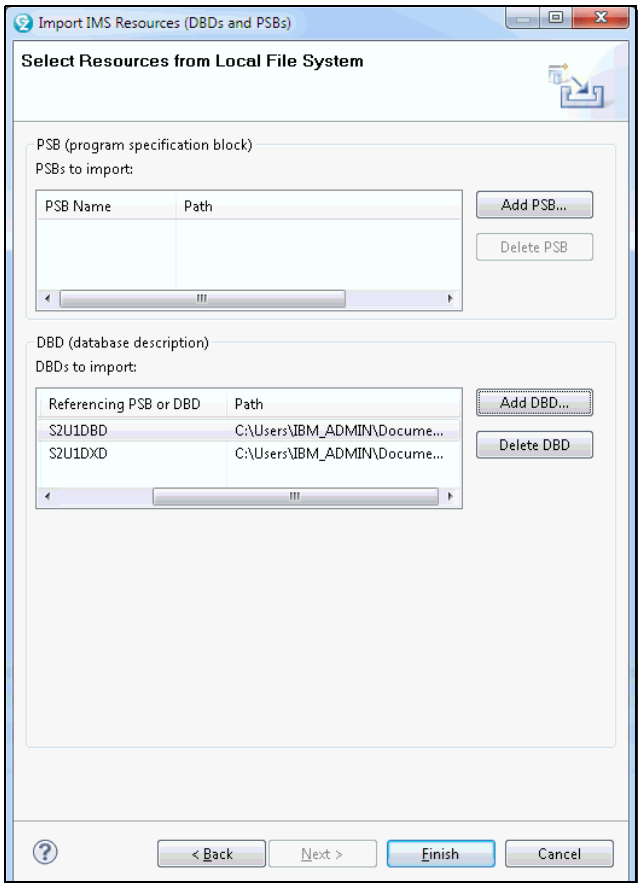


Figure 25 Keep selecting more files until you are done

As with the DBDs, a single file can contain multiple PSBs, and possibly the JCL used to generate them. IMS looks for each of the PSBs and DBDs in the file selected and adds each of them to the import selection.

Select the file that contains the PSBs that you want to import, and click **Open**. See Figure 26 on page 45.

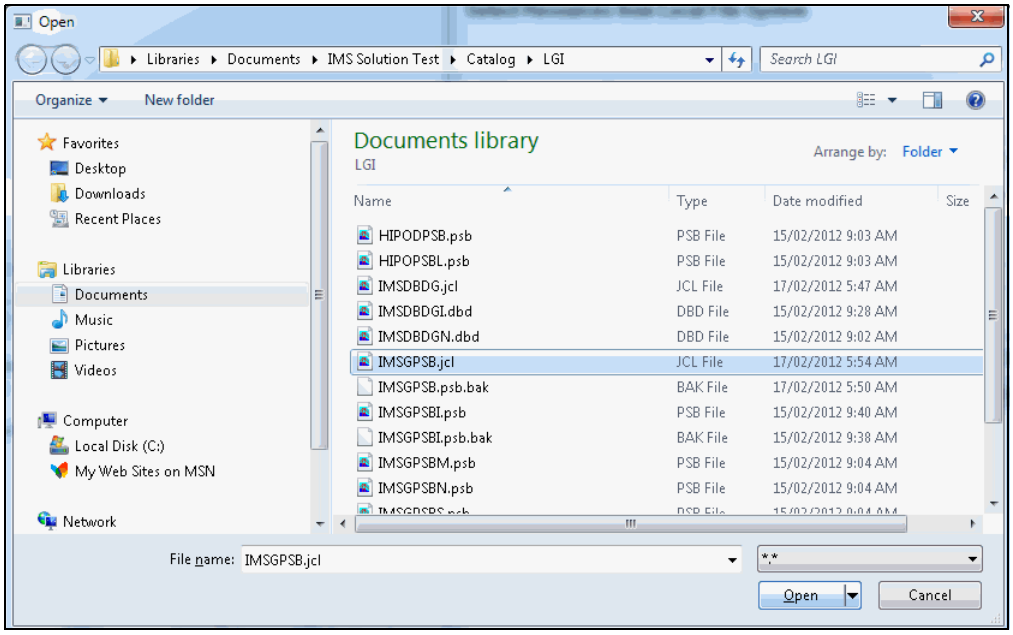


Figure 26 Selecting the PSB source to be imported

The Import IMS Resources window shows the two PSBs imported from the single file that was selected. Any of these DBDs and PSBs can now be removed from the import list by selecting **Delete DBD** and then by selecting **Delete PSB**.

The IMS Explorer also shows any DBDs that still need to be imported to satisfy the databases referenced in the PSBs to be imported. In this example, the two PSBs reference only the databases to be imported.

You are now ready to import these DBDs and PSBs into your project. As Figure 27 on page 46 shows, click **Finish**.

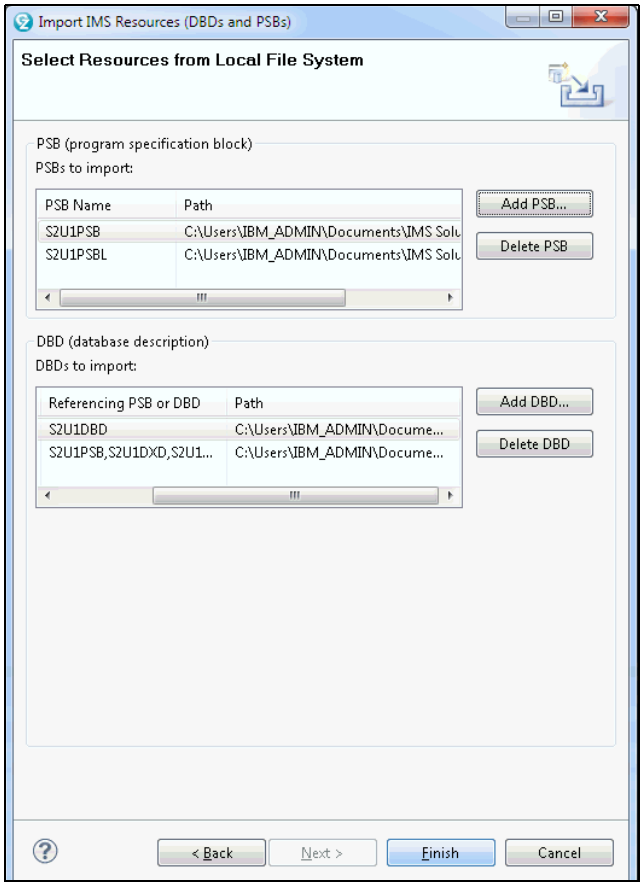


Figure 27 Lists of the DBD and PSB resources to be imported

The LGI Databases project now includes the DBDs and PSBs that you imported and several other artifacts that are needed later. At any time, you can refer to this project to see the DBD or PSB source which was imported. The project also includes the generated source for the DBDs and PSBs. At this stage, however, the generated source is the same as the imported source. The sources are the same because you did not yet add any of the metadata from the COBOL copybooks or PL/I include members. You do that soon.

To see the current layout of a database, click the arrow at the left of the DBD, to expand the list of DBDs in your project. See Figure 28 on page 47.

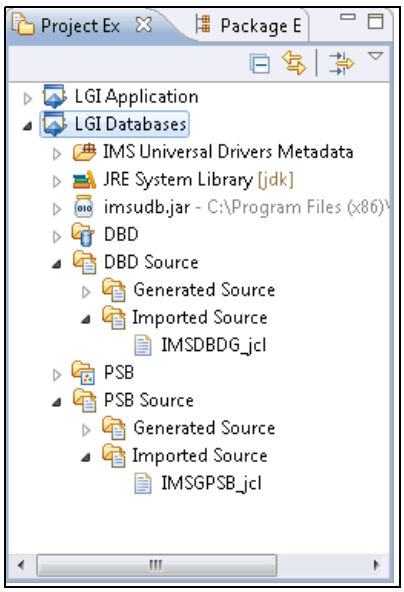


Figure 28 LGI Databases project, showing the DBD and PSB resources which are imported

You can now see the two DBDs which are imported into your project, S2U1DBD.dbd and S2U1DXD.dbd. Double-click the S2U1DBD.dbd option, as shown in Figure 29, to see the view of this database.

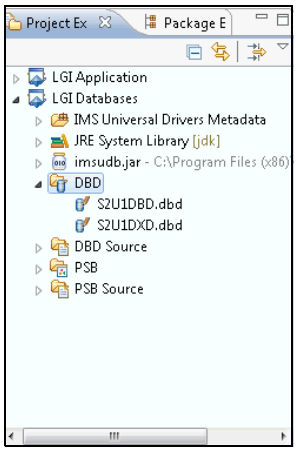


Figure 29 Expanding the DBD Resources view

You can now see the graphical layout of the database, shown in Figure 30 on page 48. The segments in the S2U1DBD database are shown. The relationships between these segments, and the fields defined to each of the segments, are also shown.

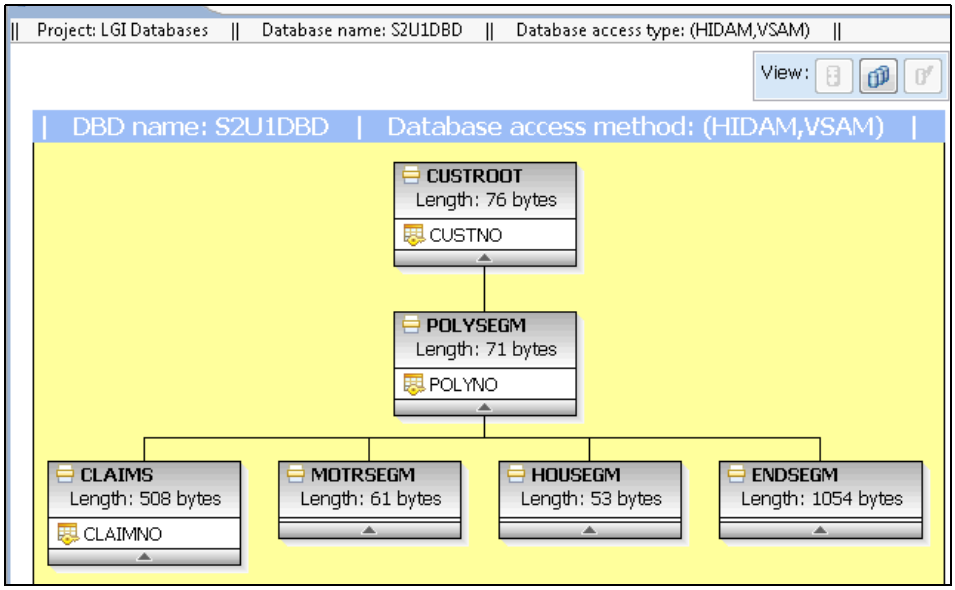


Figure 30 The Explorer view of an IMS database

From this window, you can import the rest of the metadata for this database into this IMS Explorer project. To do this import, **right-click** the CUSTROOT segment in the graphical view, and select **Import COBOL** or **PL/I Data Structures**. This action opens the window shown in Figure 31 on page 49**.**

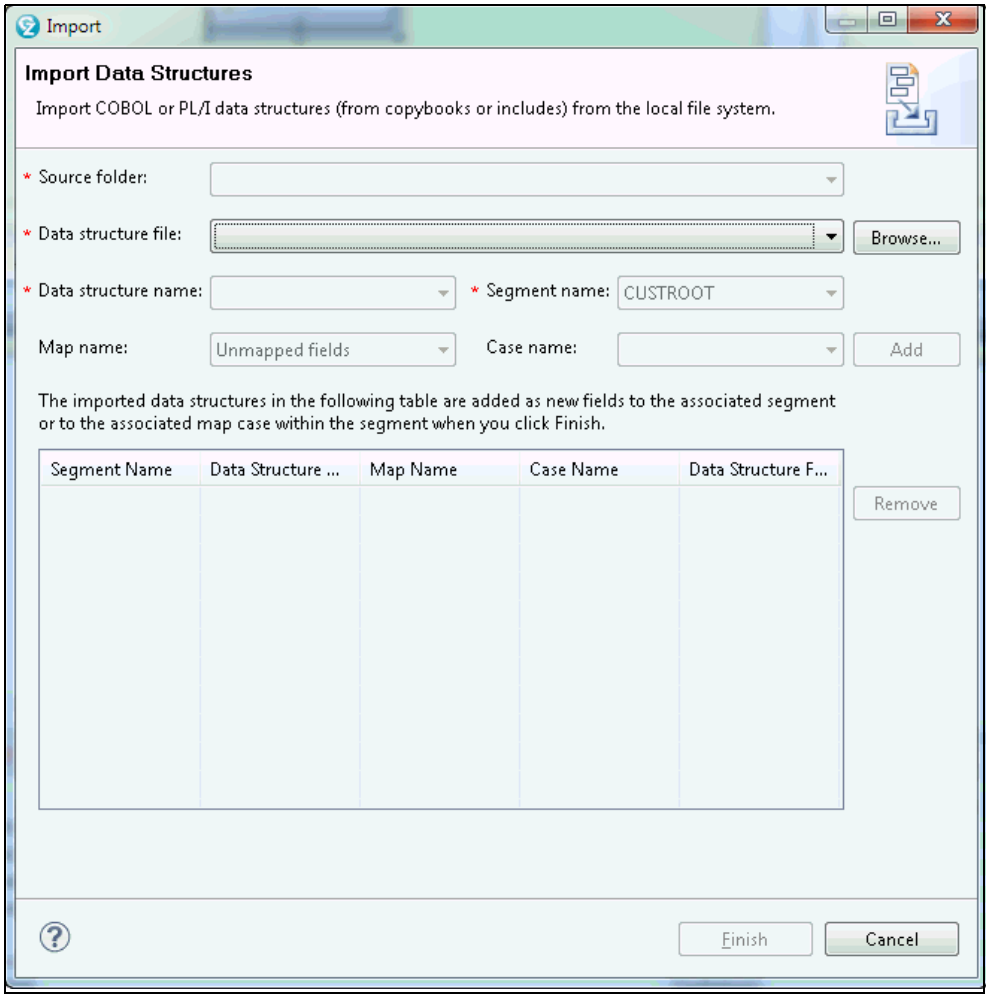


Figure 31 Importing database metadata from COBOL or PL/I

To select the file that contains the COBOL or PL/I definition for the segment, click **Browse**, as shown in Figure 31.

Select the file that contains the COBOL or PL/I definition for the custroot (client) segment, and click **Open**, as shown in Figure 32 on page 50.

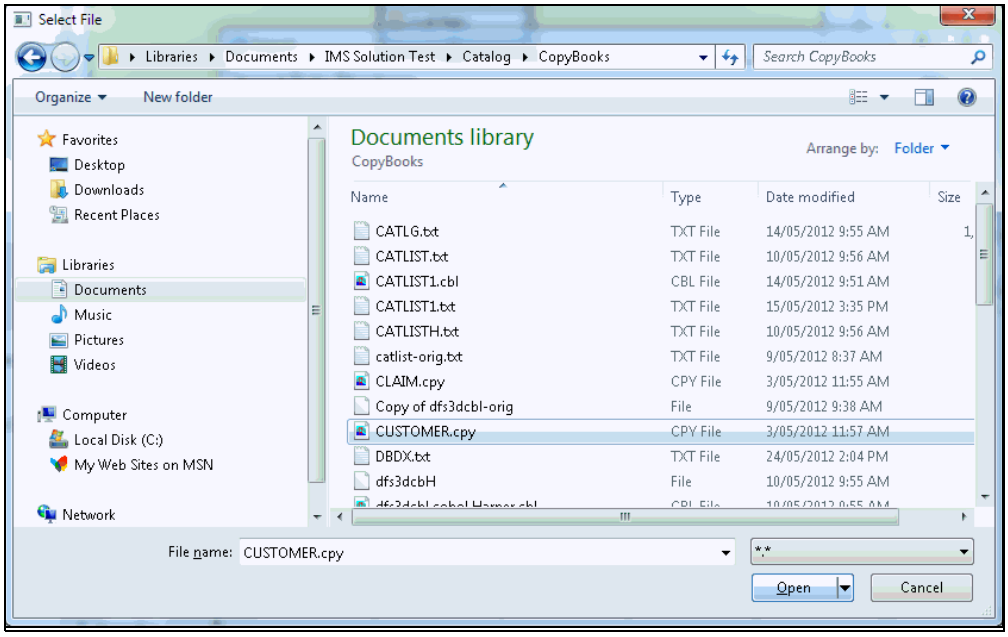


Figure 32 Select the file that contains the Copybook or Include member

Figure 33 shows the window that displays when you select Open (see Figure 32). Figure 33 shows the Import Data Structures window. This panel is where you identify the data structure to be imported for a database segment.

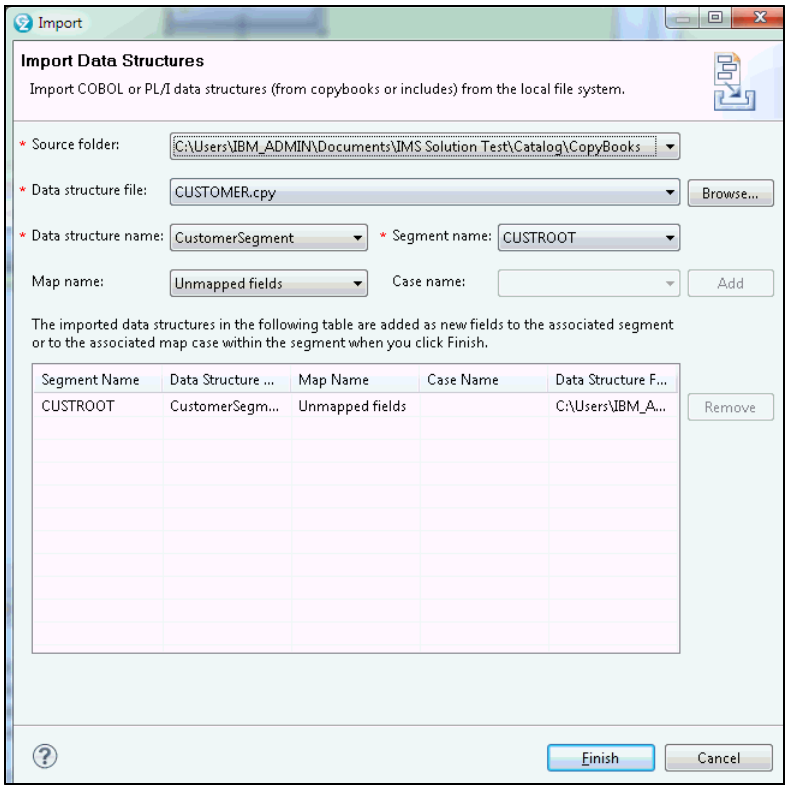


Figure 33 Identify the data structure to be imported for a database segment

In the COBOL copybook you selected, there is only one definition layout. Click **Add**, as shown in Figure 34 on page 51, to add this structure as the metadata for the CUSTROOT segment.

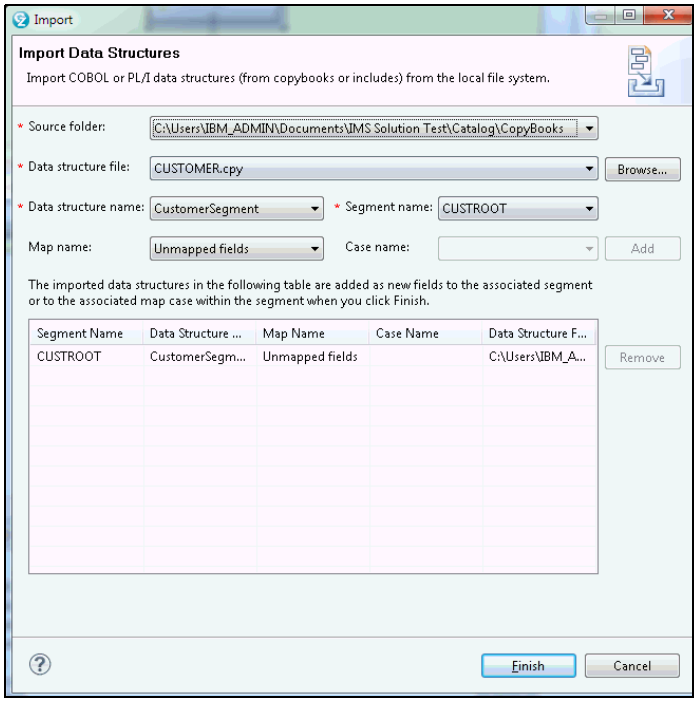
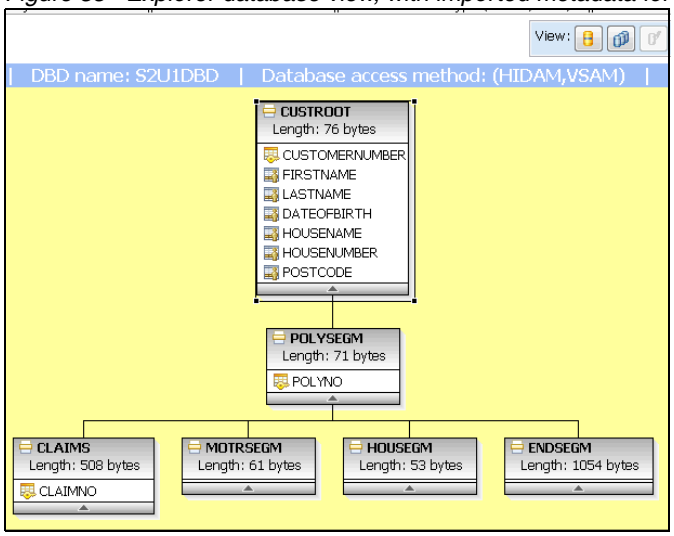


Figure 34 Importing the metadata for a database segment

From here, click **Finish** to import this segment layout into the database definition for the CUSTROOT segment. Figure 35 shows the expanded definition for this segment in the database.

Figure 35 Explorer database view, with imported metadata for one segment



The same process (right-click the segment, import the COBOL or PL/I copybook, click Add, then Finish) is used to import the metadata for each of the other segments in the database.

Figure 36 on page 52 shows the database layout with definitions for each of the segments in the database. The asterisk at the left of the database name (*S2U1DBD.dbd), indicates that the database definition is not yet saved. Select **File** → **Save** to save the enhanced database definition to the *LGI Database Project*.

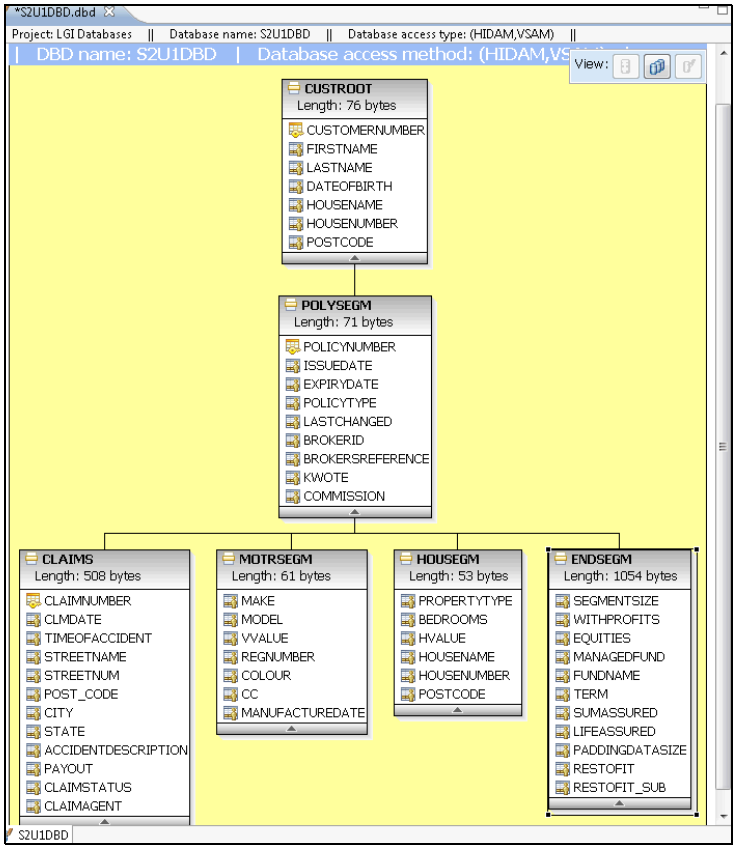


Figure 36 Explorer database view, with imported metadata for all segments

You captured the metadata for the layout of each of the segments in your database. If you click the expand arrow for DBD Source and Generated Source in your project, the `IMS catalog-enabled DBD source files` heading opens. Figure 37 shows the enhanced definitions for the databases in your project.

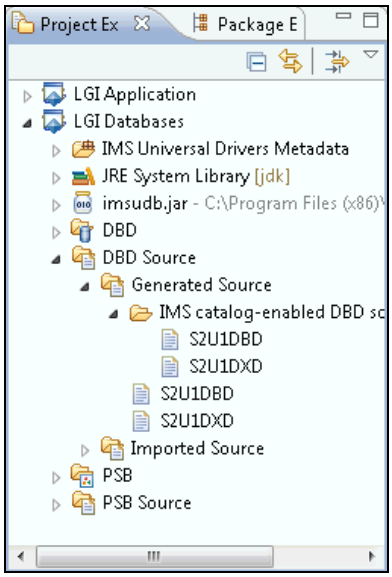


Figure 37 The catalog-enabled DBD source

You captured the expanded metadata for these databases. A snippet of the expanded DBD definition for this database is shown in Figure 38.

S2U1DBD.dbd S2U1DBD

```
DBD    NAME=S2U1DBD,                                          X
       ACCESS=(HIDAM,VSAM),                                   X
       PASSWD=NO,                                             X
       DATXEXIT=NO,                                           X
       ENCODING=Cp1047
DATASET DD1=S2U1DB,                                           X
       DEVICE=3390,                                           X
       SIZE=4096,                                             X
       SCAN=3,                                                X
       FRSPC=(0,0),                                           X
       SEARCHA=0
SEGM   NAME=CUSTROOT,                                         X
       EXTERNALNAME=CUSTROOT,                                 X
       PARENT=0,                                              X
       BYTES=76,                                              X
       POINTER=(TWINBWD)
FIELD  NAME=(CUSTNO,SEQ,U),                                   X
       EXTERNALNAME=CUSTOMERNUMBER,                           X
       BYTES=4,                                               X
       START=1,                                               X
       TYPE=C,                                                X
       DATATYPE=BINARY
FIELD  EXTERNALNAME=FIRSTNAME,                                X
       BYTES=10,                                              X
       START=5,                                               X
       DATATYPE=CHAR
FIELD  EXTERNALNAME=LASTNAME,                                 X
       BYTES=20,                                              X
       START=15,                                              X
       DATATYPE=CHAR
FIELD  EXTERNALNAME=DATEOFBIRTH,                              X
       BYTES=10,                                              X
       START=35,                                              X
       DATATYPE=CHAR
FIELD  EXTERNALNAME=HOUSENAME,                                X
       BYTES=20,                                              X
       START=45,                                              X
       DATATYPE=CHAR
FIELD  EXTERNALNAME=HOUSENUMBER,                              X
       BYTES=4,                                               X
       START=65,                                              X
       DATATYPE=CHAR
FIELD  EXTERNALNAME=POSTCODE,                                 X
       BYTES=8,                                               X
       START=69,                                              X
       DATATYPE=CHAR
```

Figure 38 Sample of a DBD with catalog-enabled source

From here, you can use existing IMS database definition processes to include the enhanced database definition into your IMS system. The following steps detail how to include the definition in your system:

- Copy the expanded DBD source to the host.

  There are several ways to copy. Recent versions of IBM Personal Communications include an FTP client with a graphical user interface (GUI). A simple file transfer is also available through your 3270 emulator.
- Generate these DBDs with standard DBDGEN processes.
- Perform an ACBGEN and Catalog Populate utility execution.

  The updated database description needs to be added into your ACB library and also to the catalog. This process can be done in a single utility execution, or separately as wanted.

The database metadata is now captured into the catalog. Existing applications continue to access their databases as they always have. New Java applications can use the catalog definitions to simplify the JDBC access to the same IMS databases.

Enhancements to the IMS Universal drivers

All Universal drivers are enhanced to use the IMS catalog and provide the following features:

- Direct access to IMS metadata in the catalog
- No longer require the separate Java metadata class
- No longer file-system dependent for metadata: Virtual deployment support
- Metadata is trusted and up-to-date
- Application Development community can access any IMS database defined in the catalog
- New complex and flexible data-type support

The following two sections describe the changes to the IMS drivers:

- Access to the IMS databases from Java
- Using the metadata information in the DL/I access

Access to the IMS databases from Java

It is required to establish a connection before any data can be accessed on DL/I databases. Traditionally, access with IMS and DLI requires a PSB; this statement is also true for Java. Traditional languages, such as COBOL and PL/I, work with a segment content, which is accessed by the SSA protocol. Java supports the following two ways to access DL/I databases through the Universal drivers (Type 4 for distributed and Type 2 for local):

- SSA oriented (for DL/I clients)
- JDBC clients

JDBC requires the additional support of metadata for the accessed information. In previous releases, this support was given by DatabaseView classes generated by the DLIModel utility. IMS 12 adds a more centralized approach by putting the metadata in a catalog. The use of metadata information offers many interesting access perspectives to the Java language (case mapping, structure selects, and so on) that are not available in other languages.

The metadata support can come from the DLIModel or the IMS catalog. This choice needs to be made at connection time, see Figure 39 on page 55.

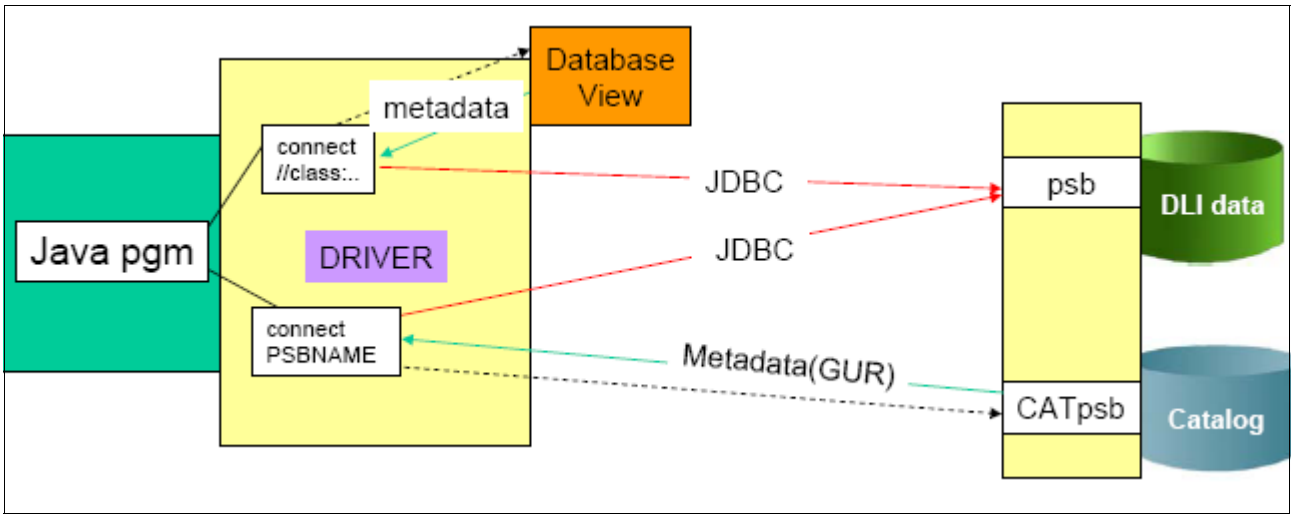


Figure 39 Connection of a Java program through the drivers

Two elements are required before you are able to access the DL/I databases:

1. A PSB (for language Java) must exist.
2. A DatabaseView class or an IMS catalog for obtaining metadata:
   - A DatabaseView class must be generated by the DLIModel utility.
   - Metadata information is available in the catalog.

Connecting to IMS for use with JDBC (JDBC client)

Example 19 shows a code excerpt of how to obtain a Type 4 connection (with an IP address and port number) with IMS for a JDBC client.

Example 19 Obtain connection with IMSDataSource for use with JDBC

```
JDBC connection:

import java.sql.Connection;
import java.sql.SQLException;
import com.ibm.ims.jdbc.IMSDataSource;
...
       IMSDataSource ds = new IMSDataSource();
       Connection conn = null;

       ds.setDatastoreName(alias);// IMS alias name defined in ODBM
       ds.setDatabaseName(psbName);
       // ds.setMetadataURL(url);---> deprecated

       ds.setUser(user);
       ds.setPassword(password);
       ds.setDriverType(driverType);

       // optional settings
       ds.setLoginTimeout(seconds);
       ds.setDescription(description);
       try {
          ds.setLogWriter(out);// set to a java.io.PrintWriter object
       } catch (SQLException e) {
          // handle exception
       }
```

```
      // JDBC type 4 driver specific settings
      if (driverType == IMSDataSource.DRIVER_TYPE_4) {
         ds.setSSLConnection(enableSSL);
         ds.setDatastoreServer(host);// IP address or DNS name of the
                        // LPAR where ICON resides
         ds.setPortNumber(port);// ICON DRDA port number
      }

      try {
         // Establish a connection using the data source
         conn = ds.getConnection();

         // do some work here

         conn.close();
      } catch (SQLException e) {
         // handle exception
      }
   }
```

Example 19 on page 55 shows the use of the IMSDatasource, which gets its properties through several setters and finally the getConnection() method to obtain the connection object.

As shown in Example 20, the setMetadataURL() method on the IMSDataSource is now deprecated and replaced by setDatabaseName(url).

The method setDatabaseName(url) sets the name of the target IMS databases to be accessed:

- If the metadata repository is the IMS catalog, the database name (url) is the name of the PSB that contains the databases to be accessed.
- If the metadata repository is the DatabaseView class, the database name is the fully qualified name of the Java metadata class that is generated by the DLIModel utility. In this case, the name must begin with *class://*.

JDBC client connection: For a JDBC client, the Connection object is the target for all other interaction with the DL/I databases.

Connecting to IMS for use as a DL/I client

Example 20 shows an excerpt of the code required to obtain access to the DL/I data from a DL/I client through the pcb object. By a DL/I client, we indicate that the access takes place as in the other languages, with PCB, SSAs, and function codes.

Example 20 Obtain a connection with PSB for use with DL/I

```
DLI connection:

import com.ibm.ims.dli.DLIException;
import com.ibm.ims.dli.IMSConnectionSpec;
import com.ibm.ims.dli.IMSConnectionSpecFactory;
import com.ibm.ims.dli.PCB;
import com.ibm.ims.dli.PSB;
import com.ibm.ims.dli.PSBFactory;
import com.ibm.ims.jdbc.IMSDataSource;
...
      IMSConnectionSpec connSpec
         = IMSConnectionSpecFactory.createIMSConnectionSpec();
```

```
PSB psb;
PCB pcb;

connSpec.setDatastoreName(alias);
connSpec.setDatabaseName(databaseName);
// connSpec.setMetadataURL(url);---> deprecated

connSpec.setUser(user);
connSpec.setPassword(password);
connSpec.setDriverType(driverType);

if (driverType == IMSDataSource.DRIVER_TYPE_4) {
   connSpec.setSSLConnection(enableSSL);
   connSpec.setDatastoreServer(host);
   connSpec.setPortNumber(port);
}

try {
   psb = PSBFactory.createPSB(connSpec);
   pcb = psb.getPCB(pcbName);

   // do some work here

   psb.commit();// or alternatively, psb.rollback()

   psb.close();
   pcb.close();
} catch (DLIException e) {
   // handle exception
}
```

DL/I connection: For a DL/I client, the PCB is the target for other interactions.

Using the metadata information in the DL/I access

With traditional languages (COBOL, PL/I, C, and so on), reading a segment brings a complete record into an I/O area, known as a *byte area*. This area needs to be interpreted. The byte area contains value fields, eventually with structures and substructures. In these languages, the extract of the elementary values, which can be in different formats (char, binary, packed decimal), is straightforward. The extract is done by overlaying the I/O area with dummy sections (COBOL copybooks, and PL/I or C include members). This technique is not available for Java.

The changes on the DBDGEN macros are explained in “DBD and PSB source changes” on page 20. The changes on DBDGEN, and the extensions to the Java access classes, allow for an enhanced access approach of the DL/I databases by Java programs. The changes on DBDGEN give the possibility to put explicit metadata information in the DBD and to store it in the catalog. Other metadata information, implicitly present in the “dummy” sections, can also be accumulated in the catalog.

Based on the exploitation of the metadata, Java (JDBC and DL/I) also offers powerful detailed and controlled access to the information in the databases. When programming Java for accessing the DL/I databases as a JDBC client or a DL/I client, you can use different styles:

- JDBC SQL approach, with or without, IBM/IMS extensions
- IBM/IMS extended DL/I style

Struct and Array objects: Traditionally, when creating a Struct or an Array object, you need to define it in a bottom-up matter that starts with the Struct attributes or the Array Elements.

IBM Extension allows for the top-down creation of a Struct or an Array object. You can define the Struct or Array object and then set the attributes and elements. It avoids potential complexity when dealing with nested structures. It also allows for setting and getting Struct and Array attributes by the attribute name.

Because of the metadata support for Java, several enhancements are available for Java. The enhancements are described in the following sections.

Variable length segment support

The IMS Universal drivers for both type-2 and type-4 database access are enhanced to support variable length database segments. Support is added to the Universal drivers:

- V11: APAR PM14766
- V12: APAR PM25951

Because the standards for JDBC do not have a concept of segment length, the default behavior is that the JDBC driver manages variable length segments internally. The behavior is managed for read, update, and insert operations, without any additional action from the client.

- SQL clients can directly read and update the length field of a variable length segment by explicitly requesting access when the connection is created.
- DL/I clients always have access to the length field.

Variable Length Segments contain a 2-byte length (LL) field that identifies the size of the segment instance. The Universal drivers are now sensitive to the LL field of a Variable Length Segment. The drivers manage the I/O area of the segment instance on standard *create, retrieve, update, and delete* calls. A new Connection Property *llField* determines whether the LL field of the segment is displayed to the program or not.

The New Connection Property llField can be set to true/false:

- llField = true

  The LL field of variable length segments is displayed to the user in both the SQL and DL/I interfaces. The user fully manages the LL field for the segment instance. The LL field is how users currently manage variable length segments.

- llField = false (default)

  The universal driver automatically manages the LL field for the user.

Variable length support: The variable length support is introduced for both DLIModel and the IMS catalog metadata.

Example 21 shows the variable length information related to the segment WARD in the DatabaseView.

Example 21 Variable segment length information in DatabaseView class

```
   static DLISegment PHDAMVARWARDSegment = new DLISegment
          (“WARD”,”WARD”,PHDAMVARWARDArray,32,900,DBType,PHAM,false);
//                                    Minlength, Maxlength
```

Example 22 shows the variable length information in the catalog.

Example 22 Variable segment length information in IMS catalog: XML

```
<segment imsName=”WARD” name=”WARD”>
   <phdam>
      <bytes minBytes=”32” maxBytes=”900”/> <======minbytes,maxbytes
   </phdam>
....
</segment>
```

Structure support

Structures are Data Structures that store a combination of values. Structure support was added specifically for the IMS catalog in IMS 12 to represent common application metadata. Example 23 represents a simple structure with the name ADDRESS-INFO, but for Java it is an object.

Example 23 COBOL structure example

```
01 PERSON.
   02 PERSNR     PIC X(6).
   02 ADDRESS-INFO.
      04 CITY    PIC X(15).
      04 STREET  PIC X(25).
      04 ZIP     PIC X(5).
```

Structure metadata is stored in the IMS catalog. Universal drivers can retrieve the metadata on Structs as XML through the GUR call. See Example 24.

Example 24 Structure information in the IMS catalog

```
..........
<field name="ADDRESS_INFO">
   <startPos>6</startPos>
   <bytes>45</bytes>
   <marshaller encoding="CP1047">
      <typeConverter>STRUCT</typeConverter>
   </marshaller>
   <applicationDatatype datatype="STRUCT"/>
   <field imsDatatype="C" name=“CITY">
      <startPos>1</startPos>
      <bytes>15</bytes>
      <marshaller encoding="CP1047">
         <typeConverter>CHAR</typeConverter>
      </marshaller>
      <applicationDatatype datatype="CHAR"/>
   </field>
   .........other fields
</field>
```

Example 25 on page 60 shows the code for the JDBC SQL retrieve of the structure (PERSON_INFO). See the COBOL definition in Example 23. In the code, you extract the subfield values from the structure object. Two approaches are shown in Example 25 on page 60.

Example 25 JDBC SQL for structure retrieve

```
   Connection conn = ....
   Statement st = conn.createStatement("SELECT * FROM”
                         + pcbName +".PERSON WHERE PERSNR = ‘XXXXXX?”);
   ResultSet rs = st.executeQuery();
   rs.next();
// 2 ways to retrieve the information
//-----------------------------------------------------------------------
// 1) standard SQL
   Struct addressInfo = (Struct)rs.getObject("ADDRESS_INFO");
   Object[] addressInfoAttributes = addressInfo.getAttributes();
   String city = (String) (addressInfoAttributes[0]).trim());
   String street = (String) (addressInfoAttributes[1]).trim());
   String zip = (String) (addressInfoAttributes[2]).trim());
//-----------------------------------------------------------------------
// 2) with IBM/IMS extensions
   StructImpl addressInfoimpl = (StructImpl)rs.getObject("ADDRESS_INFO");
   String city = addressInfoimpl.getString(“CITY”);
   String street = addressInfoimpl.getString(“STREET”);
   String zip = addressInfoimpl.getString(“ZIP”);
//-----------------------------------------------------------------------
```

With the StructImpl class, you have direct access to the structure fields with the adequate getters.

Next, new values are provided for the fields and a JDBC/SQL update stores the new structure in the DL/I database. See Example 26.

Example 26 Updating the PERSON segment with JDBC

```
   Connection conn = ....
   PreparedStatement ps = conn.prepareStatement("UPDATE " + pcbName + ".PERSON
                      SET ADDRESS_INFO =? WHERE PERSNR = “XXXXXX");
//-----------------------------------------------------------------------
// 1) standard SQL
   Object[] newAddressInfoAttribute = new Object[]
                            {“LEUVEN”,”BONDGENOTENLAAN”, “3000”};
   Struct newAddressInfoStruct = conn.createStruct(pcbName +
                            ".PERSON.ADDRESS_INFO", newAddressInfoAttribute);
   ps.setObject(1, newAddressInfoStruct);
//-----------------------------------------------------------------------
// 2) with IBM/IMS extensions
   StructImpl newaddressInfoImpl =
            (StructImpl)conn.createStruct(pcbName + ".PERSON.ADDRESS_INFO");
   newaddressInfoImpl.setString(“CITY”,”LEUVEN”);
   newaddressInfoImpl.setString(“STREET”,”BONDGENOTENLAAN”);
   newaddressInfoImpl.setString(“ZIP”,”3000”);
   ps.setObject(1, newaddressInfoimpl);
   //--------------------------------------------------------------------
   int numberOfSuccessfulUpdates = ps.executeUpdate();
```

The scenarios used in Example 26 showed code for a JDBC client. Example 27 on page 61 shows a Java code excerpt for a retrieve with a DL/I client.

Example 27 DL/I access for structure retrieve

```
   PCB pcb = ... {
   SSAList ssaList = pcb.getSSAList("PERSON");
//specify a qualified SSAList for segment PERSON = XXXXXX field PERSNR
   ssaList.addInitialQualification("PERSON","PERSNR",SSAList.EQUALS, "XXXXXX");
//Retrieve all fields from PERSON (fixed segment)
   ssaList.markAllFieldsForRetrieval("PERSON", true);
//Creates I/O area for data
   Path path = ssaList.getPathForRetrieve();
//retrieve the data
   pcb.getUnique(path, ssaList, true);
   DBStruct personinfo = (DBStruct)path.getObject("PERSON_INFO");
//-------------------------------------------------------------------
// 1)
   Object[] addressInfoAttributes = personinfo.getAttributes();
   String city = ((String)addressInfoAttributes[0]).trim();
   String street = ((String) addressInfoAttributes[1]).trim();
   String zip = ((String) addressInfoAttributes[2]).trim();
//-------------------------------------------------------------------
// 2)
   String city = personinfo.getString("CITY").trim();
   String street = personinfo.getString("STREET").trim();
   String zip = personinfo.getString("ZIP").trim();
```

Arrays

If the correct metadata is assembled in the catalog, *arrays* are also supported. Arrays are Data Structures that store a repeating combination of values. Example 28 shows an array.

Example 28 COBOL array

```
01 STUDENT.
   02 STUDENTNAME PIC X(25).
   02 AGE PIC 9(2) COMP.
   02 COURSE OCCURS 5 TIMES.
      04 COURSENAME PIC X(15).
      04 INSTRUCTOR PIC X(25).
      04 COURSEID PIC X(5).
```

Dynamic and Static Array support was added specifically for the IMS catalog in IMS 12 to represent common application metadata. Universal drivers support only Static Arrays. The drivers can retrieve the metadata on Arrays as XML through the GUR call.

A proprietary method of setting the elements within a DBArrayElementSet object is added. The method is similar to a ResultSet. It is possible to position on a specific element in an array with the following methods: next(), previous(), first(), last(), and absolute(int index).

Also, common getters and setters are provided for nested fields within an array element:

- setBoolean(String, boolean)
- setString(String, String)
- getBoolean(String)
- getString(String)
- etc.

Example 29 on page 62 shows the retrieve of the segment STUDENT, which contains an array. Notice the usage of the DBArrayElementSet class.

Example 29 Retrieve of the segment STUDENT

```
   String[] coursename =new String[5];
   String[] instname =new String[5];
   String[] courseid =new String[5];
   st = conn.createStatement();
   rs = st.executeQuery("SELECT * FROM " + pcbName +
                        ".STUDENT WHERE STUDENTNAME = XXXX” );
   rs.next()
   String studname = rs.getString(“STUDENTNAME”);
   short age = rs.getShort(“AGE”);
// 2 ways to retrieve the information
//-------------------------------------------------------------------
// 1) standard SQL
   Array fivecourses = rs.getArray("COURSE");
   Struct[] fivecourseselem = (Struct[]) fivecourses.getArray();
   // Each array element is represented as a Struct
   for (int arrayIdx = 0; arrayIdx < fivecourseselem.length; arrayIdx++) {
      Object[] courseinfo = fivecourseselem[arrayIdx].getAttributes();
      coursename[arrayIdx] = (String) courseinfo[0];
      instname[arrayIdx] = (String) courseinfo[1];
      courseid[arrayIdx] = (String) courseinfo[2];
   }
//-------------------------------------------------------------------
// 2) with IBM/IMS extensions
   int arrayIdx = 0;
   ArrayImpl fivecoursesImpl = (ArrayImpl)rs.getArray("COURSE");
   DBArrayElementSet fivecoursesArrayElementSet = fivecoursesImpl.getElements();
   try {
      while (fivecoursesArrayElementSet.next()); {
         coursename[arrayIdx] =
                        fivecoursesArrayElementSet.getString("COURSENAME");
         String instname[arrayIdx] =
                        fivecoursesArrayElementSet.getString("INSTRUCTOR");
         String courseid[arrayIdx] =
                        fivecoursesArrayElementSet.getString("COURSEID");
         arrayIdx++;
      }
   }  catch (Exception e {
      .....
   }
//-------------------------------------------------------------------
```

Example 30 shows the insert of a new STUDENT segment. The two ways of JDBC coding are shown again.

Example 30 Insert of a new STUDENT segment

```
   PreparedStatement ps = conn.prepareStatement
          ("INSERT INTO " + pcbName + ".STUDENT ("STUDENTNAME", "AGE", "COURSE")
          VALUES( ? ? ?);
   String coursenm[] = {“HISTORY”,”ENGLISH”,”FRENCH”,”ITALIAN”,”JAVA”};
   String teacher[] = {“SMITH”,”OBAMA”,”SARKOZY”,”PAOLO”,”SVLTEACHER”};
   String courseid[] = {“10”,”51”,”52”,”55”,”40”};
   ps.setString(1,”FREDERIK”);
   ps.setShort(2,41);
//-------------------------------------------------------------------
// 1) standard SQL
// fill now the ARRAY object
   Struct[] courseArrayElements = new Struct[5];
   for (int arrayIdx = 0; arrayIdx < 5; arrayIdx++) {
```

```
      Object[] courseAttributes = new Object[]
            {coursenm[arrayIdx], teacher[arrayIdx], courseid[arrayIdx]};
      courseArrayElements[arrayIdx] = conn.createStruct
            (“COURSE", courseAttributes);
   }
   Array courseArray = conn.createArrayOf(“COURSE, courseArrayElements);
   ps.setArray(3,courseArray);
//-----------------------------------------------------------------------
// 2) with IBM/IMS extensions
   int arrayIdx = 0;
   ArrayImpl fivecoursesImpl = ((ArrayImpl) ((ConnectionImpl)conn).createArrayOf
            (pcbName + ".STUDENT.COURSE"));
   DBArrayElementSet fivecoursesArrayElementSet = fivecoursesImpl.getElements();
   try {
      while (fivecoursesArrayElementSet.next()); {
         fivecoursesArrayElementSet.setString(“COURSENAME”,coursenm[arrayIdx]);
         fivecoursesArrayElementSet.setString(“INSTRUCTOR”,teacher[arrayIdx]);
         fivecoursesArrayElementSet.setString(“COURSEID”,courseid[arrayIdx]);
         arrayIdx++;
      }
   } catch (Exception e) {
   }
   ps.setArray(3,fivecoursesImpl);
//-----------------------------------------------------------------------
   int numberOfSuccessfulInserts = ps.executeUpdate();
```

Mapping support

A Map is metadata that describes how a field (or set of fields) is mapped for a particular segment instance. Metadata captures the various cases, and for each case, defines the set of fields to be used for that case. Maps can be defined to the catalog.

Maps are interpreted at run time by the universal drivers and the applicable data elements are returned based on the runtime case of the segment instance. Figure 40 shows a case selection based on the control field PolicyType.

Policy Type	Property Type	Rooms	MV alue	Address	Make	Model	Year	H Value	Color
M	-	-	-	-	Ford	Escort	1989	2K	Red
H	Single Family	5	500K	555 Disk Drive Way, 95141	-	-	-	-	-

Figure 40 Insurance segment mapped multiple ways that depend on the Policy Type control field

All case fields are displayed in the metadata. There is no concept of maps or cases at the SQL level. The support by metadata creates a requirement that all case fields have unique names from each other.

Figure 40 shows the SQLview of the data. This representation gives the impression that much space is empty on the disk. The reality is different. All maps need to be of the same length, and in this case, the physical view of the segment is similar to what is shown in Figure 41 on page 64.

Policy Type	Make	Model	Year	HValue	Color
M	Ford	Escort	1989	2K	Red

	Property Type	Rooms	MValue	Address
H	Single Family	5	500K	555 Disk Drive Way, 95141

Figure 41 Segment view on disk

Figure 41 shows that the control field PolicyType is not really a part of the mapping.

Example 31 shows an excerpt of the case mapping information in the metadata.

Example 31 Universal drivers pull metadata from the IMS catalog GUR call as XML

```
<field imsDatatype="C" imsName="POLTYPE" name="PolicyType"> <-CASE Control field
   <startPos>1</startPos>
   <bytes>1</bytes>
   <marshaller encoding="CP1047">
      <typeConverter>CHAR</typeConverter>
   </marshaller>
   <applicationDatatype datatype="CHAR"/>
</field>
<mapping dependingOnField="PolicyType">
   <case name="HOUSE"> ..............................<- CASE
      <dependingOnFieldValue valueDatatype="C" value="H"/>
      <field imsDatatype="C" imsName="PROPTYPE" name="PropertyType">
         <startPos>2</startPos>
         <bytes>15</bytes>
         <marshaller encoding="CP1047">
            <typeConverter>CHAR</typeConverter>
         </marshaller>
         <applicationDatatype datatype="CHAR"/>
      </field>
      ...
   </case>
   ...
</mapping>
```

Maps and cases are a new feature added to both the SQL and DL/I interfaces and are functionally identical. The use of case-mapping support offers the following specific create, retrieve, update, and delete behaviors:

- SQL Select and DLI Read

  When a case is not the active case that is based on the control field of the map, its fields are treated as null fields.

- SQL Insert and DLI Create

  You cannot insert values for the fields of a case unless the insert also includes the value for the control field that makes the case active.

- SQL Update/DLI Update

  You cannot update the values for the fields of a case unless the case is currently active. Or, you can update the values if the control field is also updated to a value that makes the case active.

- SQL Delete and DLI Delete

  No new behavior is observed.

Example 32 shows a case and mapping retrieve.

Example 32 Cases and mapping

```
st = conn.createStatement();
rs = st.executeQuery("SELECT * FROM " + pcbName + ".INSURANCES");
while (rs.next())  {
   byte policytype = rs.getByte("PolicyType");
   switch (policytype) {
      case ('M'):
         String make = rs.getString("Make");
         String model1 = rs.getString("Model");
         short year = rs.getShort("Year");
         int mvalue = rs.getInt("MValue");
         String color = rs.getString("Color");
      case ('H'):
         String propertyType = rs.getString("Make");
         short room = rs.getShort("Room");
         int hvalue = rs.getInt("HValue");
         String address = rs.getString("Address");
   }
}
```

Redefines

Redefines are overlapping fields. Redefines are a way of remapping a field in the database. See Example 33.

Example 33 COBOL structure example

```
01 PERSON.
   02 ADDRESS PIC X(45).
   02 ADDRESS-INFO REDEFINES ADDRESS. <-------------------
      04 CITY PIC X(15).
      04 STREET PIC X(25).
      04 ZIP PIC X(5).
```

Example 34 shows the way this redefine information is stored in the catalog.

Example 34 Redefine metadata information in the IMS catalog

```
<field imsDatatype="C" name="ADDRESS">
   <startPos>1</startPos>
   <bytes>45</bytes>
   <marshaller encoding="CP1047">
      <typeConverter>CHAR</typeConverter>
   </marshaller>
   <applicationDatatype datatype="CHAR"/>
</field>

<field name="ADDRESS_INFO" redefines="ADDRESS> .............< redefine
   <startPos>1</startPos>
```

```
   <bytes>45</bytes>
   <marshaller encoding="CP1047">
      <typeConverter>STRUCT</typeConverter>
   </marshaller>
   <applicationDatatype datatype="STRUCT"/>
   ...
</field>
```

Fields that overlap can be interchangeable. The search performance of a query depends on the type of field in the qualification statement:

- Key fields: Fastest
- Searchable fields
- Not searchable (that is, application defined fields)

The universal drivers promote a field upwards when issuing queries against IMS.

The following example shows how the key field, KEY; and the non-key field, NONKEY; redefine each other:

```
SELECT * FROM TBL WHERE NONKEY=A ===>becomes
SELECT * FROM TBL WHERE KEY=A
```

The trace and log files show the promotion in the SSA list that is sent from the universal driver to IMS.

Public converter interfaces

Public interfaces are added to the internal type converters of the universal drivers for clients to use in implementing their own type converter routines. Support is added in IMS 10, IMS 11, and IMS 12.

The new ConverterFactory class allows users to create the following converter interfaces:

- DoubleConverter
- FloatConverter
- IntegerConverter
- LongConverter
- PackedDecimalConverter
- ShortConverter
- StringConverter
- UByteConverter
- UIntegerConverter
- ULongConverter
- UShortConverter
- ZonedDecimalConverter

The converter classes contain a *getter and setter method* for converting the data type to a binary representation.

The IMS catalog is built to store information about the fields with a user-defined type. This defined type allows users to interpret binary data stored in IMS in whatever form is required for their application.

Example 35 on page 67 shows the IMS catalog metadata that contains a user-defined type.

Example 35 Field with Fully Defined Converter Class

```
<field name="PACKEDDATEFIELD">
   <startPos>40</startPos>
   <bytes>5</bytes>
   <marshaller encoding="">
      <userTypeConverter>class://com.ims.PackedDateConverter</userTypeConverter>
      <property name="pattern" value="yyyyMMdd"/>
      <property name="isSigned" value="N"/>
   </marshaller>
   <applicationDatatype datatype="OTHER"/> <===== set to Other ====
</field>
```

User Type converters must extend the com.ibm.ims.dli.types.BaseTypeConverter abstract class.

Universal drivers include a User Defined Type Converter along with its source for the PackedDate type, which stores date information as a PackedDecimal field. See Example 36.

Example 36 User Defined Type Converter for the PackedDate type

```
public class PackedDateConverter extends BaseTypeConverter {

   public Object readObject(byte[] ioArea,int start,int length,Class objectType,
          Collection<String> warningStrings) throws ConversionException {
       ...
   }

   public void writeObject(byte[] ioArea, int start, int length, Object object,
          Collection<String> warningStrings) throws ConversionException {
      ...
   }
}
```

Application-transparent metadata access

Users can explicitly issue a GUR call through the following method, which is only available for DL/I clients:

```
PCB.getCatalogMetadataAsXML(String resourceName, byte[] resourceType)
```

The following input parameters exist:

- resourceName

  The name of the PSB or DBD resource to be retrieved.

- resourceType in the format PCB.PSB_RESOURCE or PCB.DBD_RESOURCE

  Identifies whether a PSB or a DBD resource is to be retrieved.

The call returns an XML document that contains the metadata for the resources requested. The XML document conforms to the IMS managed schemas. The following schemas are made available with the IMS catalog:

- DFS3XDBD.xsd
- DFS3XPSB.xsd

Example 37 on page 68 shows the explicit GUR call.

Example 37 Example of an explicit GUR call

```
byte[] gurOutput = pcb.getCatalogMetadataAsXML("STLIVP1", PCB.PSB_RESOURCE);
```

Example 38 shows the *getCatalogMetaDataAsXML* public method definition.

Example 38 Example of a getCatalogMetaDataAsXML public method definition

```
    /**
     * This method returns a byte array containing the requested catalog resource
     * as an XML document.
     * <p>The following code fragment illustrates how to retrieve the timestamp
     * (TSVERS) value from the IMS Catalog.
     * <blockquote>
     * <pre>
     * PCB pcb = psb.getPCB("DFSCAT00");
     * SSAList ssaList = pcb.getSSAList("HEADER", "DBD");
     * Path path = ssaList.getPathForRetrieveReplace();
     * pcb.getUnique(path, ssaList, false);
     * String timestamp = path.getString("TSVERS");
     * </pre>
     * </blockquote>
     *
     * @param resourceName the name of the PSB or DBD resource in the catalog
     * @param resourceType the type of resource (PCB.PSB_RESOURCE or PCB.DBD_RESOURCE)
     * @param timestamp the TSVERS version for the resource following the
     *        pattern yyDDDHHmmssff
     * @return the resource metadata in XML
     * @throws DLIException if the resource was not found in the catalog or an error
     *         occurs during processing
     * @see #PSB_RESOURCE
     * @see #DBD_RESOURCE
     */
    public byte[] getCatalogMetaDataAsXML(String resourceName,
                  byte[] resourceType, String timestamp)
                  throws DLIException
;
```

Recommended maintenance

When a new function becomes available through the maintenance stream, it is important to be current.

This section summarizes the recent maintenance for IMS 12, which provides and enhances the IMS catalog support. The list of authorized program analysis reports (APARs) in Table 1 on page 69 represents a snapshot of the current maintenance at the time of this writing. Therefore, the list becomes incomplete (or even incorrect) at the time of reading. Ensure that you contact your IBM Service Representative for the most current maintenance at the time of your installation. Also check on IBM RETAIN® for the applicability of these APARs to your environment and to verify prerequisites and postrequisites.

Table 1 on page 69 shows a list of IMS 12 APARs for catalog support.

Table 1 IMS 12 APARs for catalog support

APAR no.	Area	Text	PTF and notes
PM36434	Catalog	Preconditioning code for IMS 12	UK78043
PM38214	IVP	Support for the IMS catalog	UK78071
PM38939	Catalog	Preconditioning code for IMS 12	UK78042
PM38942	Catalog	Preconditioning code for IMS 12	UK78028
PM42903	Catalog	Preconditioning code for IMS 12	UK78067
PM42904	Catalog	Preconditioning code for IMS 12	UK78066
PM42905	Catalog	Preconditioning code for IMS 12	UK78045
PM42906	Catalog	Preconditioning code for IMS 12	UK78070
PM42908	Catalog	Preconditioning code for IMS 12	UK77993
PM42909	Catalog	Preconditioning code for IMS 12	UK78069
PM45935	IMS Universal drivers	Support for the IMS catalog	UK77995
PM55836	Open Database Manager	ABEND0C4	UK77127
PM61228	Advanced ACBGEN	Support for the IMS catalog	UK78428
PM62879	Catalog Populate Utility DFS3PU00	Support for the IMS catalog (performance)	UK79622
PM63976	ODBM	Excessive TCB attach and detach processes when running ODBM with RRS=N	OPEN
PM63977	ODBM	Excessive TCB attach and detach processes when running ODBM with RRS=N	OPEN
PM64745	High Performance Pointer Checker	Support for the IMS catalog	UK79246
PM65139	SSA	SSA enhancement for IMS 12	OPEN
PM68881	Index Builder	Support for the IMS catalog	PACKAGING
PM68396	DBD/PSB/ACB	Support for the IMS catalog	INTRAN
PM69378	IMS Universal drivers	SSA qualification enhancement for IMS Universal drivers to allow searching on application defined fields	OPEN
PM69550	Advanced ACBGEN	Support for the IMS catalog	OPEN
PM70701	Authorized JSCBAUTH	Support for the IMS catalog	OPEN

IMS catalog support: PM42909 (UK78069) for IMS catalog support changed the length (INQELEN in the DFSINQY mapping macro) of the AIB INQY ENVIRON I/O area from 100, to 108 bytes, by adding the catalog indicator. This increase in length requires a change and a recompile for programs that use the INQY ENVIRON call.

The supporting documentation for the IMS catalog function is available online in the IMS Version 12 library section of the IBM Information Management Software for z/OS Solutions Information Center. See this website:

http://publib.boulder.ibm.com/infocenter/imzic

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at the IBM International Technical Support Organization (ITSO), San Jose Center.

Geoff Nicholls is a Consulting IT Specialist, working in the IMS Solution Test team for the IBM Silicon Valley Laboratory, and is based in Melbourne, Australia. Before his current role, Geoff was a member of the Worldwide IMS Advocate team for 12 years, providing Consulting, Education, and Services to clients in many industries around the world. Geoff is the co-author of 12 IMS IBM Redbooks® publications. Before joining IBM, Geoff worked as an Applications Programmer and Database Administrator for several insurance companies and another mainframe vendor. Geoff has a Bachelor of Science, majoring in Computer Science, from the University of Melbourne.

Paolo Bruni is an Information Management software Project Leader with the IBM ITSO, since 1998. He is based in the Silicon Valley Lab, San Jose. Paolo authors IBM Redbooks publications about IMS, DB2® for z/OS, and related tools and conducts workshops worldwide.

Dougie Lawson is a Senior Software Specialist in IMS with IBM Global Services in the United Kingdom. He works in software support for the UK and Ireland and with the European software support teams. He has 30 years of experience in the IT field and 29 years working with IMS. His areas of expertise include IMS, DB2, Linux, and z/OS. Before he joined IBM in 1994, he was working as a Systems Programmer for a large United Kingdom bank, responsible for IMS and DB2 systems and related products.

Egide Van Aershot holds an Engineering degree in Electricity and Nuclear Physics from the University of Leuven, Belgium. He joined IBM in 1967 and was responsible for many computer installations related to teleprocessing and database management in Belgium. In 1997, he moved from IBM Belgium to IBM France, where he worked as an Architect and Consultant at the IBM Program Support Center in Montpellier. Since 1997, he specialized in Java, service-oriented architecture, IMS and IBM WebSphere® applications, mainly on z/OS systems, and participated in many projects related to the Internet. Egide is co-owner of the patent “Methods, systems, program product for transferring program code between computer processes.” Currently, Egide is a contractor for Zinteg C.V. He teaches IBM System z WebSphere Application Server and WebSphere MQ classes for IBM in Northern Europe.

Thanks to the following people for their contributions to this project:

Kyle Charlet
Nathan Church
Catherine Cox
Ben Johnson
Jeff Fontaine
Hiroaki Katahira
Susan Kimura
Charles Ling
Rick Long

Victor Sze
Richard Tran
William Li
IBM Silicon Valley Lab

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and client satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-4812-00 was created or updated on September 20, 2012.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks