

**IBM Information Management Software**



**Geoff Nicholls  
Paolo Bruni  
Dougie Lawson  
Egide Van Aershot**

# IMS 12: The IMS Catalog

## Introduction

In this document, we provide a description of the 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, such as business impact analysis.

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

In this paper, we discuss the following topics:

- ▶ 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

## Overview and objectives of the catalog

Prior to the introduction of the IMS catalog, information about the structure of the DL/I databases was spread across the following different data structures:

- ▶ The data base description (DBD)

The DBD defines the characteristics of a database, such as the database's organization and access method, the segments and fields in a database record, and the relationship between 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 indices.

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

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

- ▶ The program specification block (PSB)

PSBs are used to define two things - the view of the databases to be used by a program, as well as any logical message destinations. These views are called program communication blocks (PCBs) as 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 the program can access are described, and 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, and the programs view of the hierarchical structure of the database can be different to the hierarchical structure defined in the DBD.

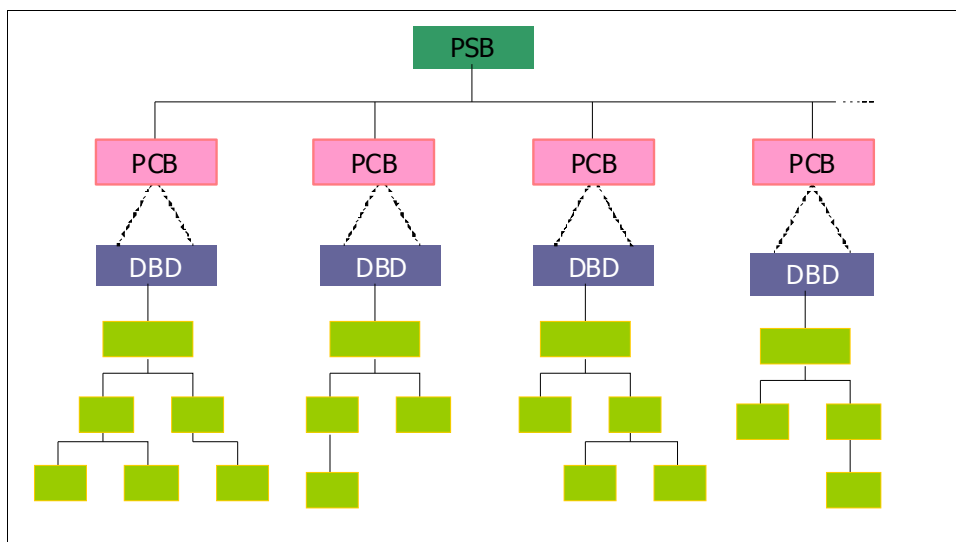


Figure 1 A Program Specification Block (PSB) with multiple Program Communication Blocks (PCBs)

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 using DBD, PSB, and ACB libraries is proprietary. Typically, a DBD only defines a subset of the fields in a database record - the sequence (key) fields and any that are needed by a secondary index. Other fields in a record would only be defined in a COBOL copybook or 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, the contents of ACBLIB are now offered as a Java class.

With the move to making IMS data more widely and easily accessible outside the mainframe, application programmers need to have an easy and consistent access to the metadata. Since IMS V8, this data was provided using the DLIMODEL utility. This utility generates Java class files that contain a static definition of the database design as 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, and 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 should 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 using the JDBC drivers and is also available to any tool or application. IMS metadata is available in an 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. Hence the data is always current, as the consumers get metadata dynamically from the active IMS catalog HALDB database rather than from static class files.

The catalog in IMS 12 also includes a versioning system, allowing the current version and a user-specified number of previous versions of the metadata 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 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, to keep the catalog in a trusted state at all times after initialization. The IMS catalog is designed to be 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 timestamps 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 (high availability large database) database: a partitioned, IMS full-function database type. Prior to 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 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 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 has traditionally been held in the DBD and PSB libraries, along with additional information that is enabled by the IMS 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 30).

### Segments of the catalog database

The segments in the catalog are:

▶ Resource header (Item name and type) for each DBD and PSB

▶ DBD resource

- Database structure definitions (ACCESS, RMNAME, etc.)
- 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 resource

- Program Control 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)

<pre> 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 0 0 0 0 0 0 0 D D FFF SSS C D D 0 0 0 0 0 0 0 0 D D F S C D D 0 0 0 0 0 0 0 0 D D F S S C C D D 0 0 0 0 0 0 0 DDDD F SSS CCC DDDD 000 000 000                 </pre>												
<pre> 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                 </pre>												
SEG-NAME	SC#	LV	PAR	-LEN-	---FREQ---	C P L L P P L L E	RULES	PHYS.	SEG-NAME	D-B-NAME	FORM	LOG CHLD
HEADER	*	1	1	0	56	0.00	XX	X				
DBD	*	2	2	1	552	0.00	XX X	X				
CAPXDBD	.	3	3	2	32	0.00	X X	X				
DBDRMK	.	4	3	2	264	0.00	XX X	X				
DSET	.	5	3	2	96	0.00	XX X	X				
DSETRMK	.	6	4	5	264	0.00	XX X	X				
AREA	.	7	3	2	40	0.00	XX X	X				
AREARMK	.	8	4	7	264	0.00	XX X	X				
SEGM	+	9	3	2	376	0.00	XX X	X				
CAPXSEGM	.	10	4	9	32	0.00	X X	X				
SEGMRMK	.	11	4	9	264	0.00	XX X	X				
FLD	"	12	4	9	904	0.00	XX X	X				
FLDRMK	.	13	5	12	264	0.00	XX X	X				
MAR	"	14	5	12	704	0.00	XX X	X				
MARRMK	.	15	6	14	264	0.00	XX X	X				
PROP	"	16	6	14	304	0.00	XX X	X				
LCHILD	+	17	4	9	72	0.00	XX X	X				

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		VAR LEN					
												*PFX*	LEN= 22	MAX= 48	PFX+MAX= 70			



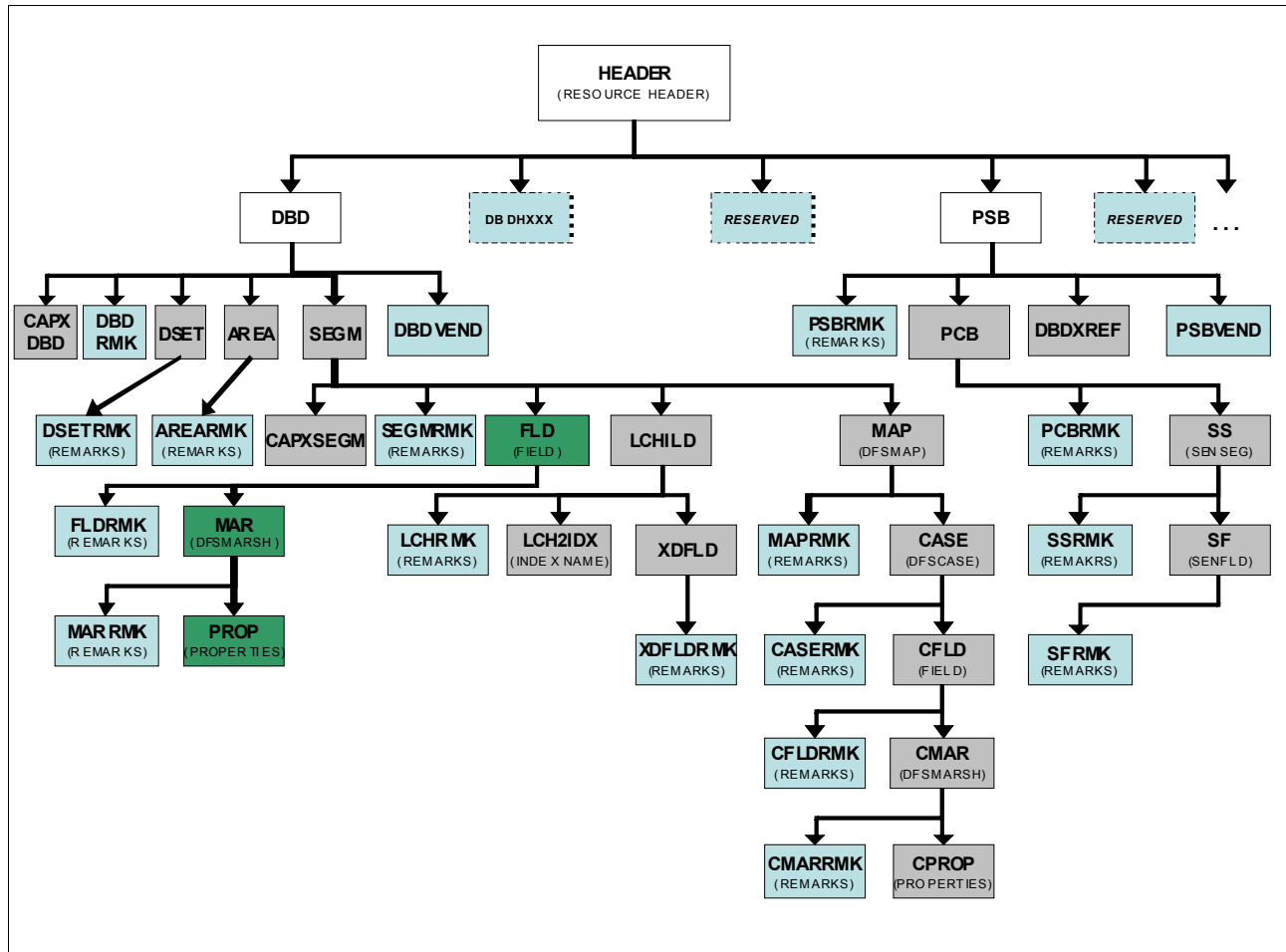


Figure 2 Database structure for the catalog Database, DFSCD000

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 some reserved segments in the database to allow for future expansion.



Most of segment types (such as DBD, DSET, AREA, SEGM, FLD,LCHILD, XFLD) contain metadata about the specific element that they represent, with 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 would normally require each HALDB database to be registered in DBRC. Since there are companies who choose not to use DBRC in some of their environments, 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
- ▶ Alias and sharing
- ▶ Definitions needed for 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 life cycle.

### Installation

The first step of 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, but the object code in SDFSRESL should 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)
```

---

Once the DBDs and PSBs have been copied to your libraries, you need to build them as ACBs 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 can be done in three ways.

- ▶ The database data sets can be allocated with the JCL similar to that shown in Example 4.
- ▶ The IMS Catalog Partition Definition Data Set utility (DFS3UCD0) can be used to create the catalog partition definition database dataset if DBRC is not being used. It also creates the database datasets for the catalog database.
- ▶ The IMS Catalog Populate utility (DFS3PU00) is used to load or insert records into the catalog database, whether or not 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 PROCLIB member.

*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))

---

Once 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 whether you choose to use:

- ▶ A unique IMS catalog for each system,
- ▶ A unique DFSDfxxx member for each system,
- ▶ A shared IMS catalog, or
- ▶ 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 “Alias 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, 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

Once you have defined the IMS catalog database data sets, you need to 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 have been rebuilt from DBD and PSB libraries using the type-1 ACB generation utility for the new release before the ACBLIB is used, including populating the catalog. The catalog populate process should only need to be done once for each system using that IMS catalog.

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

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

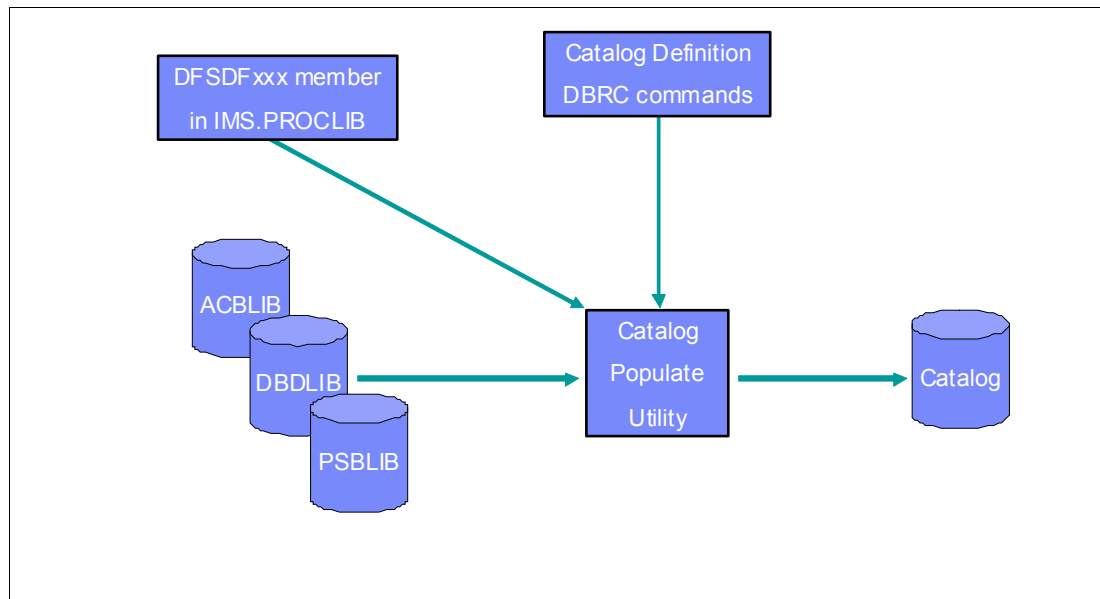


Figure 3 IMS Catalog Populate utility (DFS3PU00)

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 GSAM DBDs, as 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, since at this stage the IMS catalog only holds the rudimentary information that is available from the DBD and PSB definitions. See “The role of the IMS Enterprise Suite Explorer for Development” on page 30 for details on this process.

Example 7 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(DFSVMDB),DISP=SHR
  
```

```
//IMSACB01 DD DSN=IMS12.IMS12D.ACBLIBA,DISP=SHR
//IMSACB02 DD DSN=IMS12.IMS12D.ACBLIB.DOPT,DISP=SHR
```

## ACB generation and changes

Once 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 timestamps to ensure consistency, so you can ensure the data in the catalog can always be trusted as an accurate equivalent of the metadata 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, as 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 new 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.

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

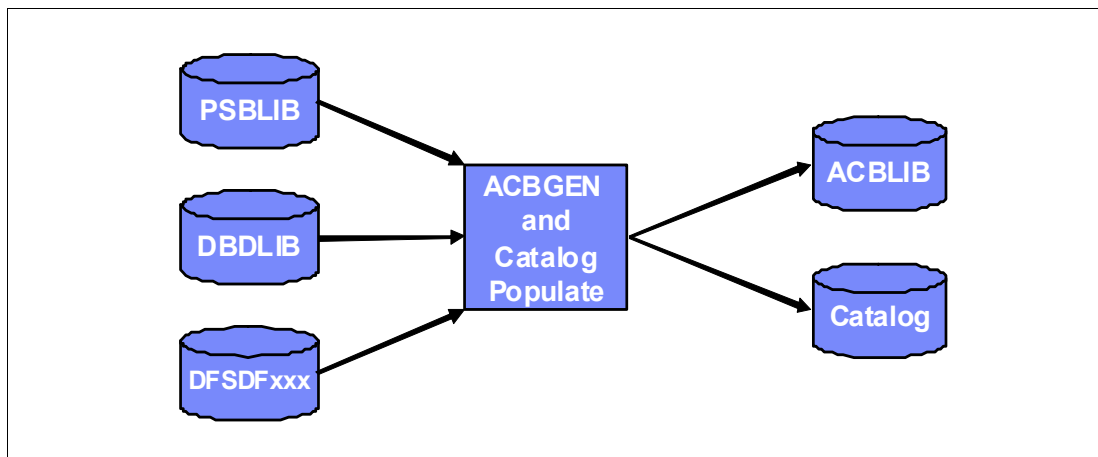


Figure 4 ACB generation and IMS catalog populate utility

The new ACB generation utility writes logs and if the IMS catalog is defined in DBRC then it updates the RECON. It can run as a BMP or DLIBATCH utility. If run as a BMP, be careful to check the catalog database is open by IMS for update access. Typically, IMS opens the catalog database in read-only mode.

## IMS Catalog Copy utility

As soon as you have migrated to an IMS catalog, you have to ensure the IMS catalog is updated every time you update an IMS ACB library. When migrating applications from one environment to another, from test through to production for example, the new 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 and copies any included ACB, DBD, and PSB libraries from the export data sets into their destination data sets.

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

If 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 updates to 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 being 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 IMS Catalog Copy utility runs in analysis-only mode, the report reflects only the potential statistics if the IMS catalog were loaded or updated from the ACB libraries currently being 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 timestamped, and IMS uses these timestamps 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/or the minimum retention period for which information is to be kept in the catalog. This is similar to the way 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 two parameters 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, default value is 2. When the maximum value is reached, the oldest version (based on ACBGEN timestamp) 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, default value is 0 (function disabled). When a version of the catalog metadata is older than the specified, it becomes candidate for removal 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 has not expired then a new catalog record will be kept, the oldest will not be 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 should still have the previous version available.

For logically related databases, where the past the DBD information has been kept only in the DBDLIB and not the ACBLIB, the catalog populate utility sets the timestamp to zero until the combined ACBGEN/Populate utility has been run for those databases.

Metadata information in the catalog can be removed by the IMS Catalog Record Purge utility (DFS3PU10) if outdated or the version has exceeded the specified retention value. When using DFS3PU10, the default of 2 versions applies. If no retention value is specified, each execution of ACBGEN or Populate Utility will add a new 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 timestamp) 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, default 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 would require 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.

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

The backup, logging, and recovery of the IMS catalog is the user's responsibility; this could be done by running stand-alone MVS™ utilities or by DFSMSshm backup and recovery. In some cases using the IMS catalog populate utility to rebuild the database may 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 a 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 SYSIN data set to define the data that you would normally use on INIT.DB and INIT.PART if the database were registered in DBRC (see Example 10). 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,
      DSNPREFIX=IMSTESTS.DFSCD000,
      KEYSTHEX=FFFFFFFFFFFFFFFFFFFFFFFFFFFF)

HALDB=(NAME=DFSCX000,HIKEY=YES)

PART=(NAME=DFSCX000,
      PART=DFSCX01,
      DSNPREFIX(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 in Example 12. TYPE=CATDBDEF is a new specification just for the un-registered IMS catalog database.

*Example 12 DFSMDA definition*

---

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

---

## Alias and sharing

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

The way ACB libraries in IMS 10 or IMS 11 are used depends on whether you are using local online change, global online change and member online change. 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, dynamically allocating ACB libraries and using member online change.

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

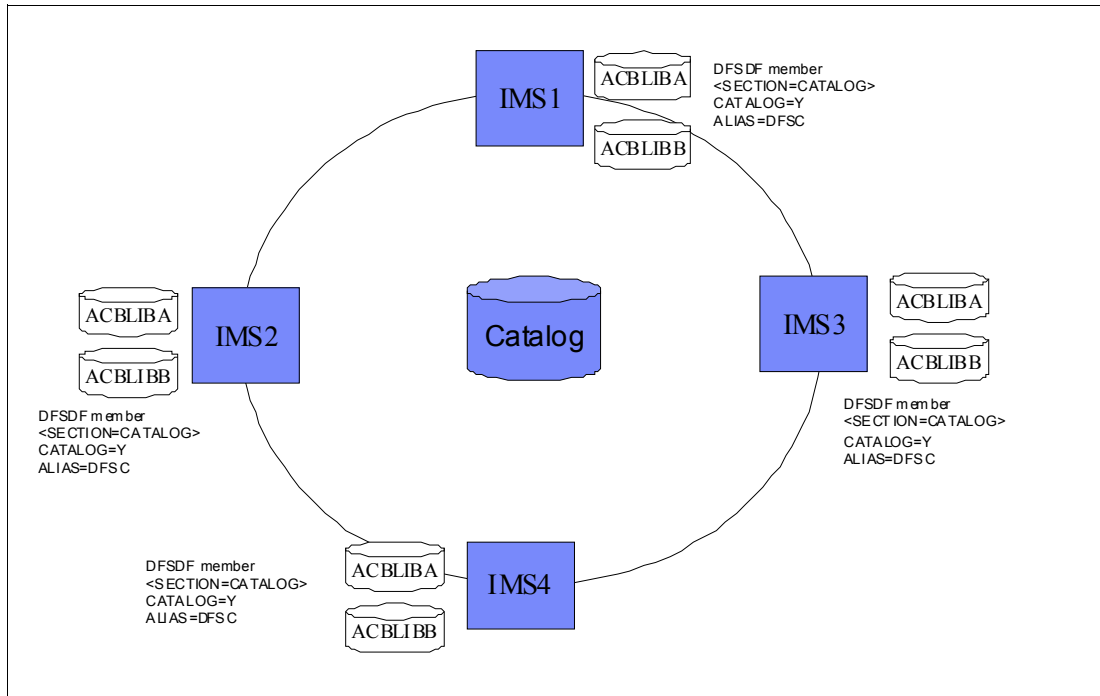


Figure 5 Multiple IMS, cloned ACBLIB, shared IMS catalog

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

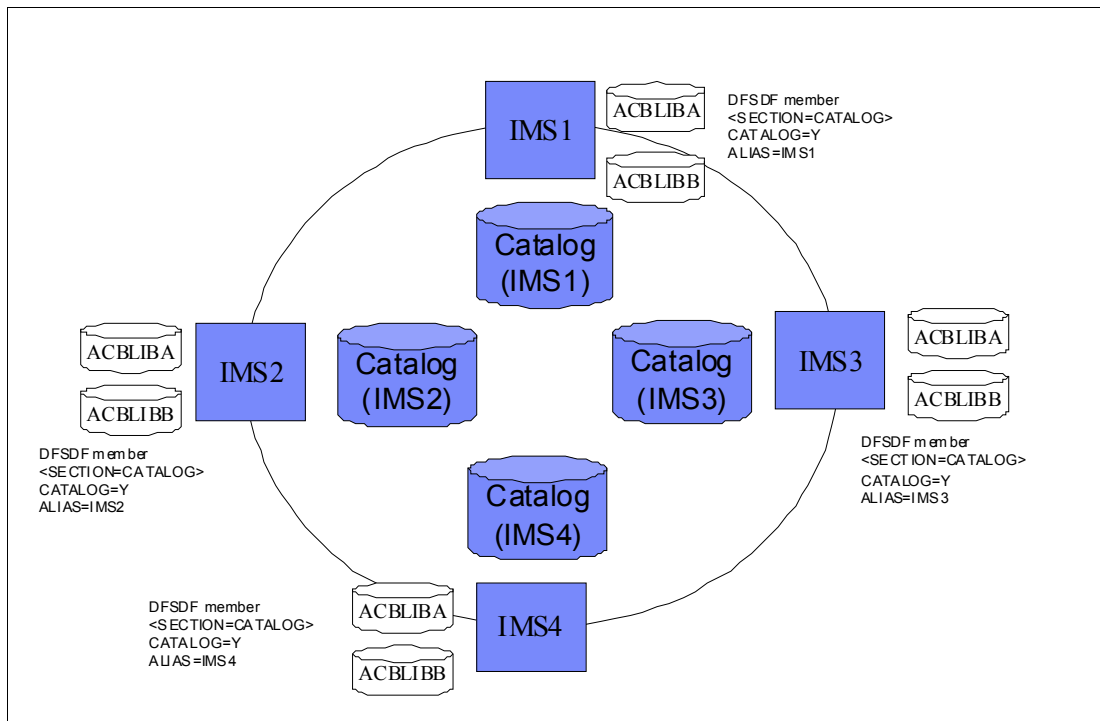


Figure 6 Multiple IMS, cloned ACBLIB, one IMS catalog per IMS

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.

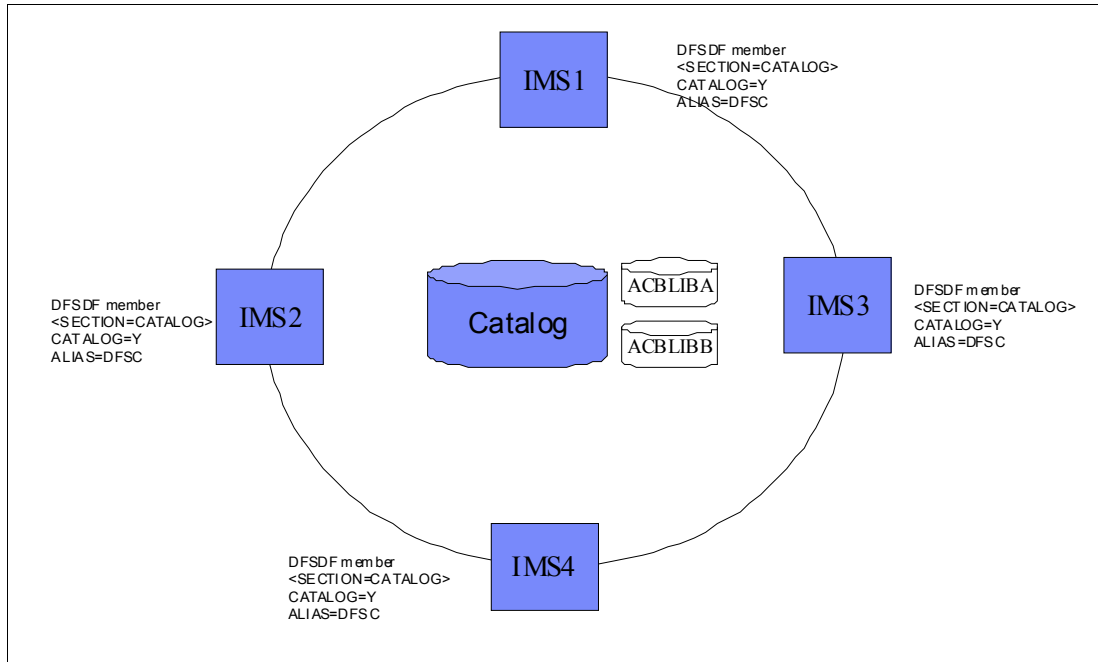


Figure 7 Multiple IMS, shared ACBLIBs, shared IMS catalog

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

## Definitions needed for IMS catalog

Before you can have the IMS catalog created and initial 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 using the recommended ALIAS name.

### Example 13 Single IMS system

---

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

---

For example if you have two or more IMS systems using shared queues you may 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 using the IMS ID and 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. It 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 are the DBD 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, and as a result several DBDGEN and PSBGEN macros have been enhanced with IMS 12.

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

- ▶ DBD and PSB source changes
- ▶ Get Unique Record (GUR) 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. A lot 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 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 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 are:

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 a 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 timestamp Java data types.

The PATTERN parameter applies only when DATE, TIME, or TIMESTAMP is specified on the DATATYPE keyword in the FIELD statement and CHAR is 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 given segment instance, as shown in Figure 8.

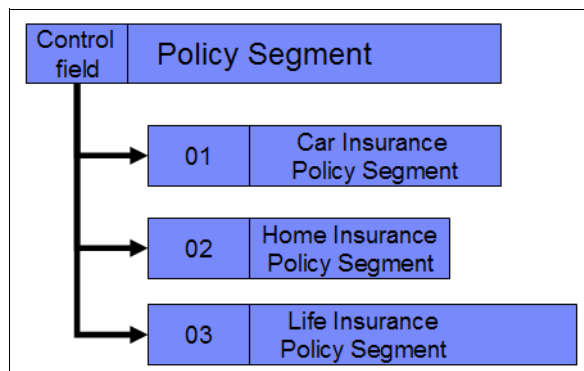


Figure 8 Several mappings for same segment

► 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 given segment instance. If the control field does not contain a value that corresponds to a CASEID in a DFSCASE statement for this map, then this map is not being 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 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 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 given 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 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 to 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 used to describe a database definition with IMS12.

#### ***Changes to the DBD statement:***

► ENCODING

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

► EXTERNALNAME

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

**Changes to the *FIELD* statement:**

► NAME

Specifies the name of this field within a segment type, 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.

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

► CASENAME

Name of the map case to which this field belongs.

Required only to associate a field statement with the preceding DFSCASE statement that defines the map case.

Must match the value specified on the NAME parameter of the DFSCASE statement.

► DATATYPE

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, XML

► REDEFINES

Field that specifies the name of another field in the segment, must be the same length as the field that is being redefined.

Field cannot be an ARRAY or contain an ARRAY.

► EXTERNALNAME

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

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

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

► MAXOCCURS

For DATATYPE=ARRAY only, a required numeric value specifying the maximum number of elements in an ARRAY.

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

► MAXBYTES

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 possible sum 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 a structure that contains a nested field that is defined as a dynamic array.

► STARTAFTER

– Specifies the name of the field that directly precedes this field in the segment. 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 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

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.

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 allows the user to specify comments that are stored in the DBDLIB, PSBLIB, ACBLIB and IMS catalog. This helps 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 (GUR) 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 DB or PSB from the catalog database. The GUR call functions similarly to a Get Unique (GU) call followed by a series of Get Next within Parent (GNP) calls, to read all the segments in this database record. It can only be used to retrieve the metadata definition of an IMS database or PSB from the catalog database.

The GUR call reads an entire database record from the catalog database, and returns an XML document containing 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 by passing back a token returned by the initial GUR call. The data is buffered by IMS on the assumption the entire XML document will be needed to minimize the overhead 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 re-read the catalog database. This new call is designed to simplify access to the database definition metadata, and minimize the overhead when retrieving it.

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

DFSDDLTO and IMS 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 IMS catalog. When the I/O area is too small for output then output buffer is kept and a token is returned in the AIB to allow application to resume copying data from buffer to I/O area on subsequent calls.

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

The GUR call can only be used to the IMS catalog database, and 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 timestamps from the ACBLIB and stores them into the database control blocks. The GUR call uses the timestamp to find the currently active member used by IMS or the first record only when IMS doesn't have a timestamp. IMS does not have a timestamp for resources not defined to this IMS.

### Segment Search Arguments (SSAs) 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 shows an application program issue the GUR call, to retrieve the metadata for the S2U1DBD database.

*Example 15 Sample Application using 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 AIBRSV4         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 above are shown in Example 16. Note that 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
tatatype="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 only be specified up to the DBD or PSB level in the catalog database

No more than two SSAs:

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

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

*Example 18 DFSDDLTO 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 IMS catalog, a PSB to access the catalog database is automatically attached to the PSB loaded for the application. IMS can then use that to access the metadata in the IMS catalog.

### Direct catalog interface

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

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

## Indirect catalog interface

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

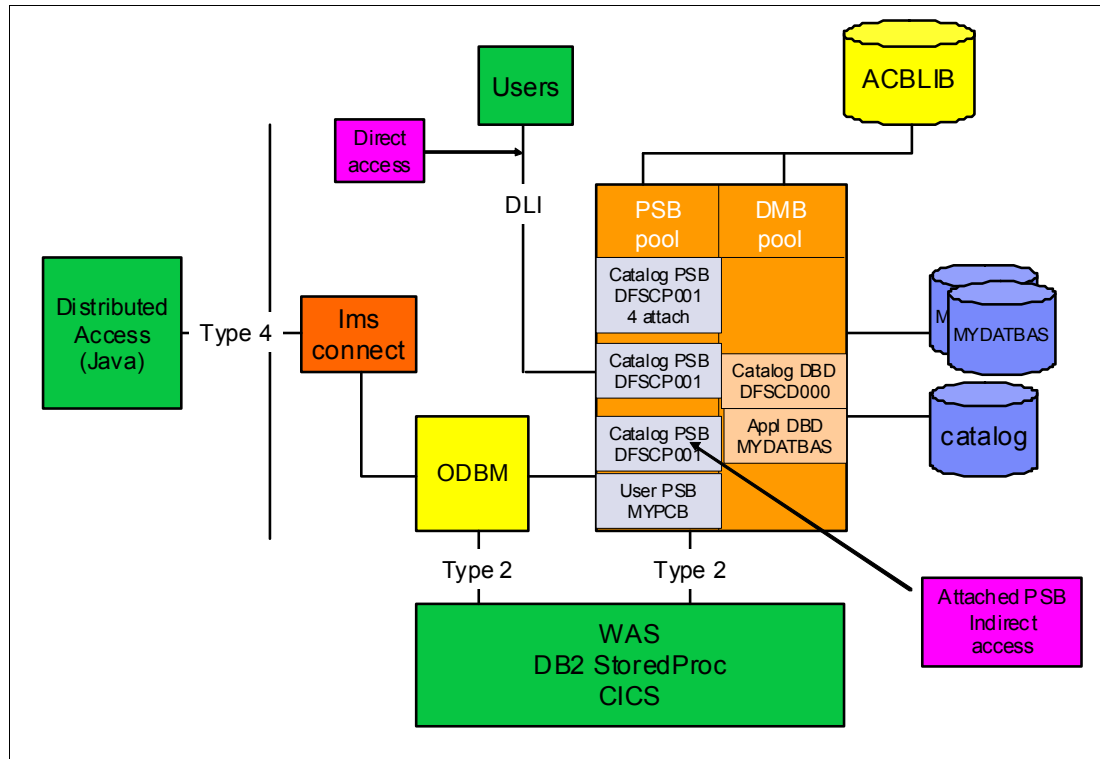


Figure 9 IMS catalog access

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 MPP applications (running in an online IMS), the dynamic attach of a PSB containing 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. IMS catalog database availability doesn't affect application availability: if the IMS catalog is offline you could get an abend (U3303) but that will not cause the application to terminate.

In a DL/I batch environment, the dynamic attach of the IMS catalog PCB occurs during batch initialization. Scheduling is only done once for 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 new searching by offset and length instead of field name; fields are no longer required to be defined in the DBD.

- Support is added for DFSDDL0 and IMS REXX

- ▶ Performance is the same as non key field search
- ▶ IMS scans the database looking for matches
- ▶ SSA contains offset and length followed by operator and value

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

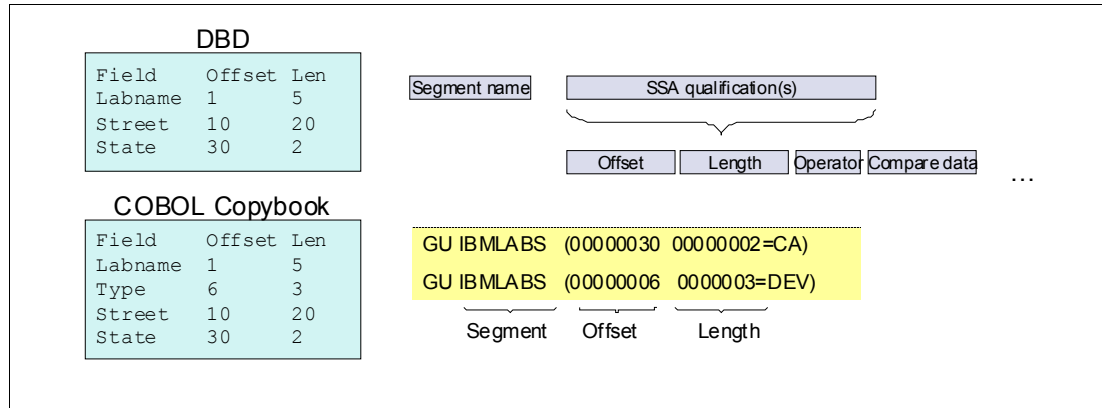


Figure 10 Using get by offset

Field level sensitivity allows user to change layout of the segment 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 for the fields "labname" and "type" is valid.

```
GU IBMLABS*0 (LABNAME =SVL &00000006 00000003=DEV)
```

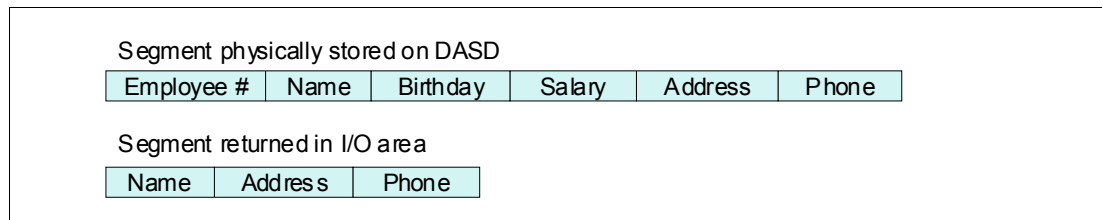


Figure 11 Field sensitivity

In this request SSA, offset is relative to the physical starting position in the segment visible to your program. Should you attempt to search for fields that are outside the sensitive area (that is, past the end of the segment), IMS returns an ‘AK’ status code. If no segment is 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 such as updating IMS database and program definitions, and using standard SQL to manipulate 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 stored in IMS by using standard SQL.

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

The population of the IMS catalog is mainly done from PSB and DBD sources. You have seen that 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 C or PLI includes. After updating in the Explorer the metadata information in the PSBs and DBDs, those sources can be sent to the host for the GEN process which will cause the metadata to be populated into the IMS catalog.

In Figure 12, you can see how the Explorer is able to interact with the IMS catalog directly via a type 4 driver connection, and also indirectly via FTP import and export.

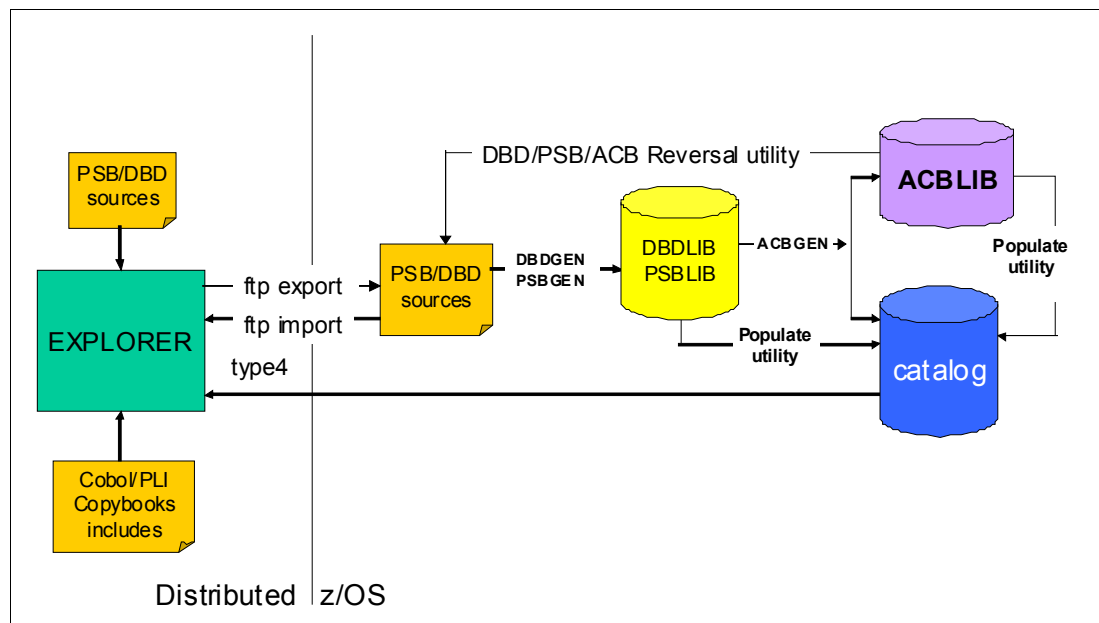


Figure 12 Explorer and catalog

Direct update from the explorer is not available, an intermediate ACBGEN with population of the catalog is required.

DLIModel utility is a helpful tool for visualizing, documenting all details about DBDs and PSBs sources, and required to produce the “com.ibm.ims.db.DLIDatabaseView class” for accessing IMS databases from Java applications on remote and local locations via the type 4 and type 2 drivers. Both 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 in the IMS Enterprise Suite Explorer. The IMS Enterprise Suite Explorer uses the centralized IMS catalog on z/OS®. The metadata currently available in the DatabaseView class, is integrated in the IMS catalog and many more details are kept. The concept of the IMS catalog makes the metadata more dynamic and shared and its implementation avoids the need for distributing the class to all sites where it should be used in Java programs with DL/I (also via JDBC) access thru the Universal Drivers type 2 and type 4. The IMS database drivers type 2 and type 4 have been adapted to integrate this dynamism.

In summary the Enterprise Explorer:

- Incorporates DLIModel functionality

- Support for migration of DLIModel projects is provided.
- ▶ Provides 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 Explorer

In the previous section we explained how the input was taken from local PSB and DBD sources.

The explorer can take input in other ways:

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

The IMS catalog is the final location where all information on 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 has to be sent back to the consolidating host by FTP. Figure 13 shows the Export panel.

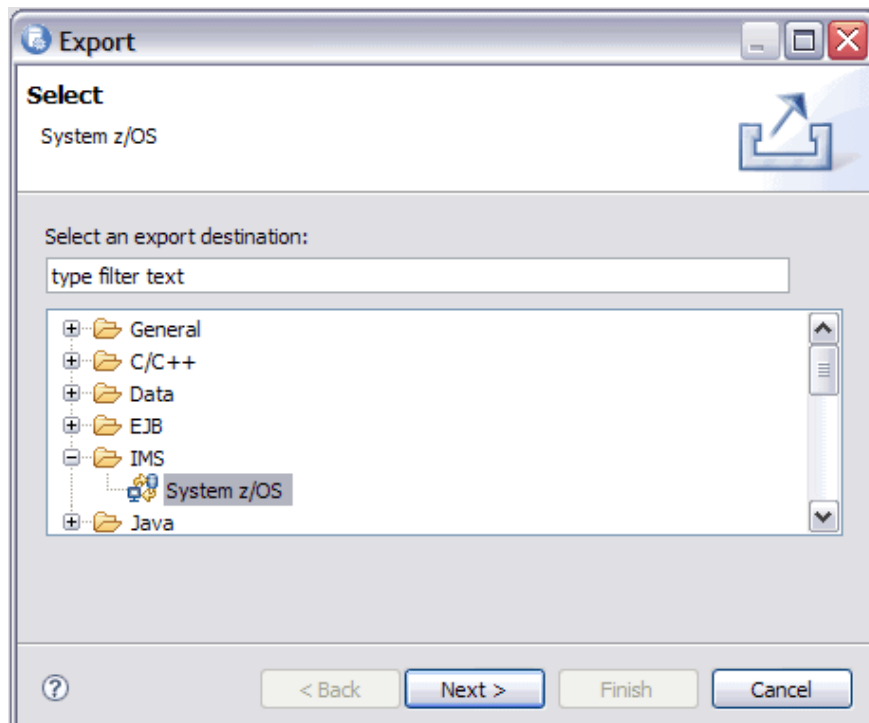


Figure 13 Export to z/OS host with FTP

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



## 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, PSBLIB), is available to users in the IMS catalog.

The explorer is an important tool for exploiting and updating the information in the catalog. The ES Explorer can pull information from the catalog database, work with it, update it and finally send it back to 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 DBD and PSB sources
  - Via type-4 driver direct access to catalog
- ▶ To z/OS
  - FTP export to DBD and PSB sources

## Extending IMS database definitions with the IMS Explorer

In this section, we take you through the extension of an IMS DBD with the IMS Explorer for Development. Using the IMS Explorer, we can quickly and easily add metadata information to IMS databases, using the COBOL Copybooks or PL/I Include members 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 could previously be 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 COBOL copybooks, and extend the DBD ready for generation on the host system.

Before starting, you need to install 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 run-time versions of this code, and the one needed for this task is the “shell-sharing” version. This installs with the IBM Installation Manager as an extension to IBM Rational® Developer for zSeries® or IBM Rational Developer for zEnterprise®.

To start, run Rational Developer for zEnterprise (or Rational Developer for zSeries), and open the IMS Explorer perspective as shown in Figure 14.

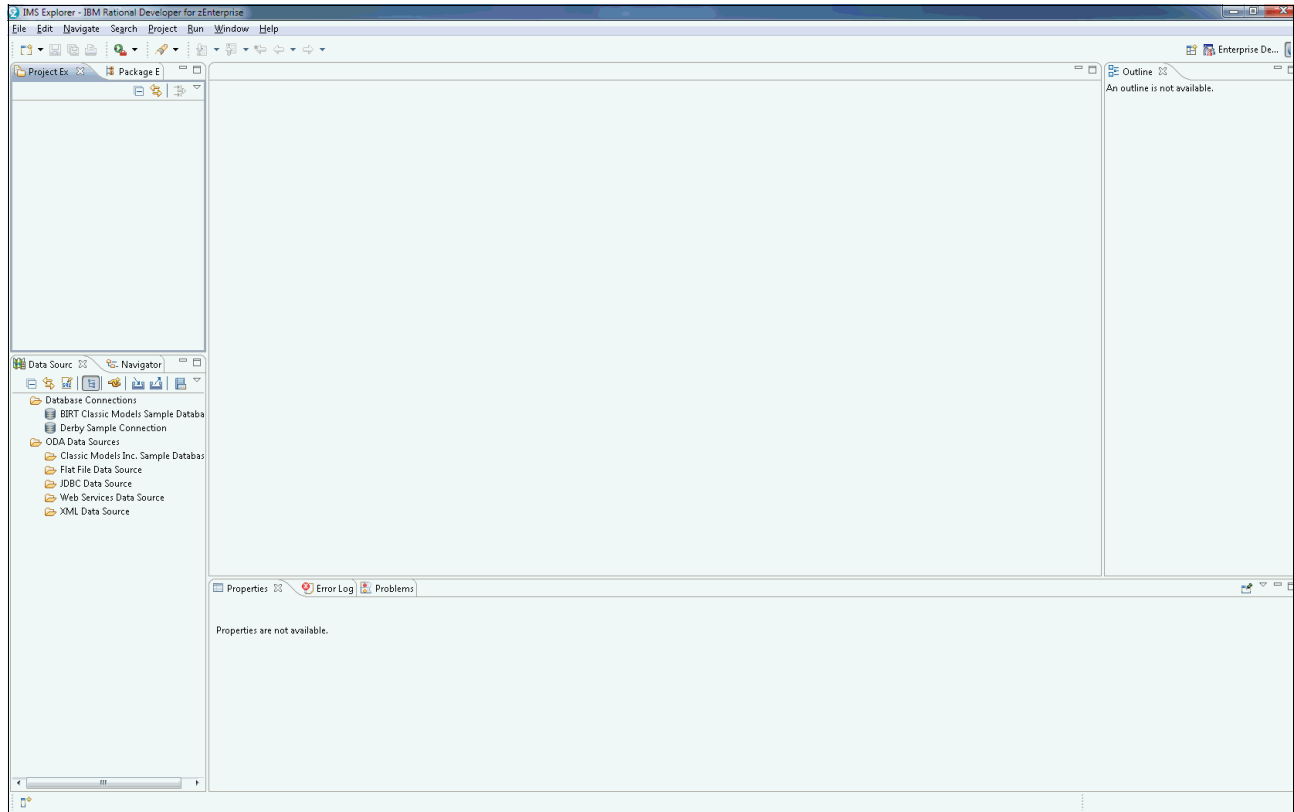


Figure 14 IMS Explorer perspective in Rational Developer for zEnterprise

To create a new IMS Explorer Project, right-click in the Project Explorer window (top-left), and select **New -> Project...** to bring up the create new project wizard (Figure 15).

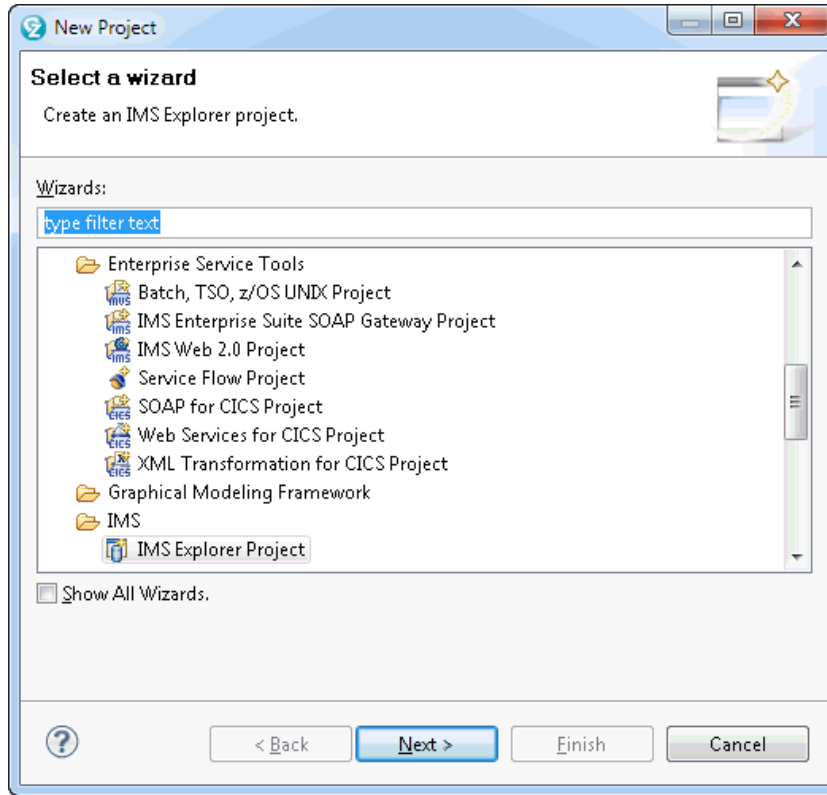


Figure 15 Create a New Project wizard

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

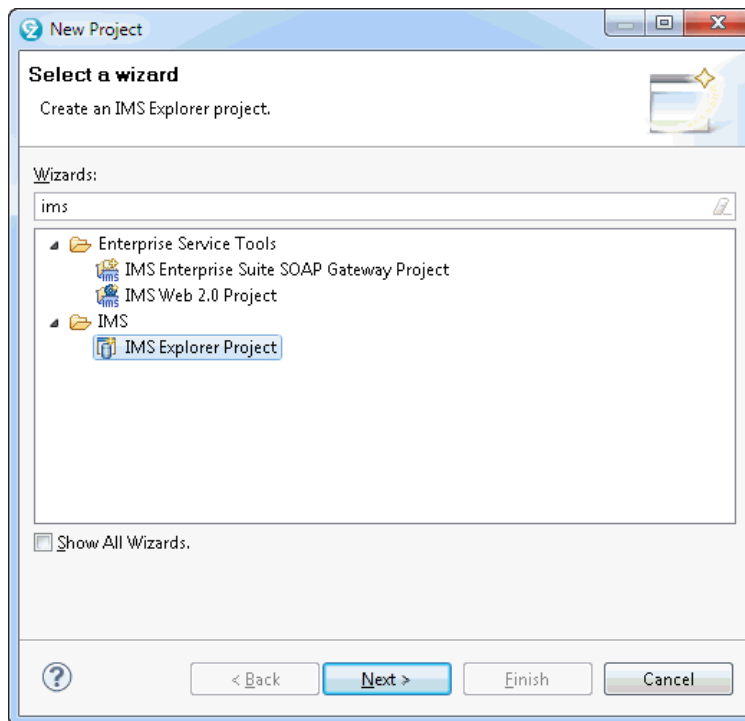
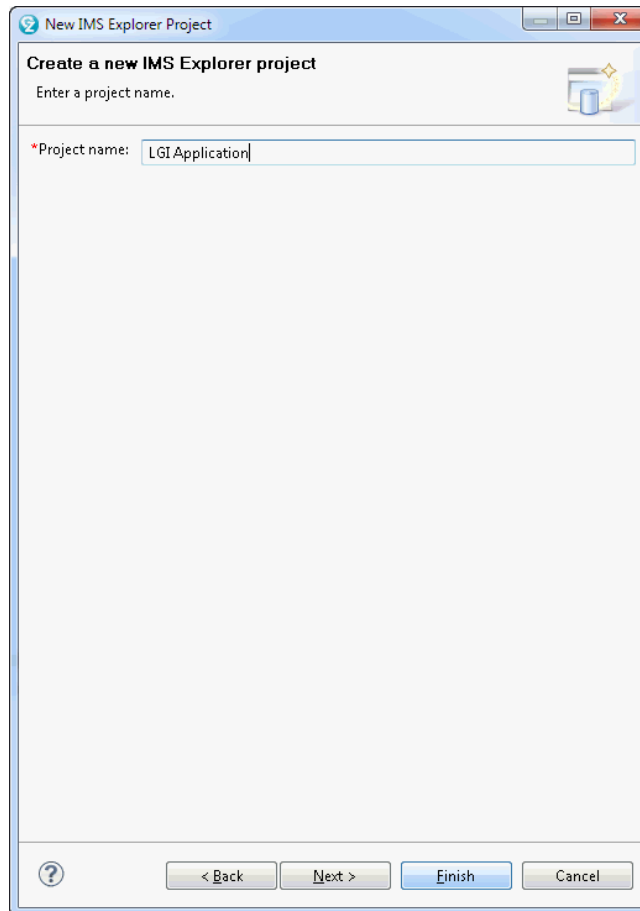


Figure 16 IMS Explorer Project

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



*Figure 17 Name the IMS Explorer Project*

When the wizard completes, RDz shows the named project in the Project Explorer window (see Figure 18).

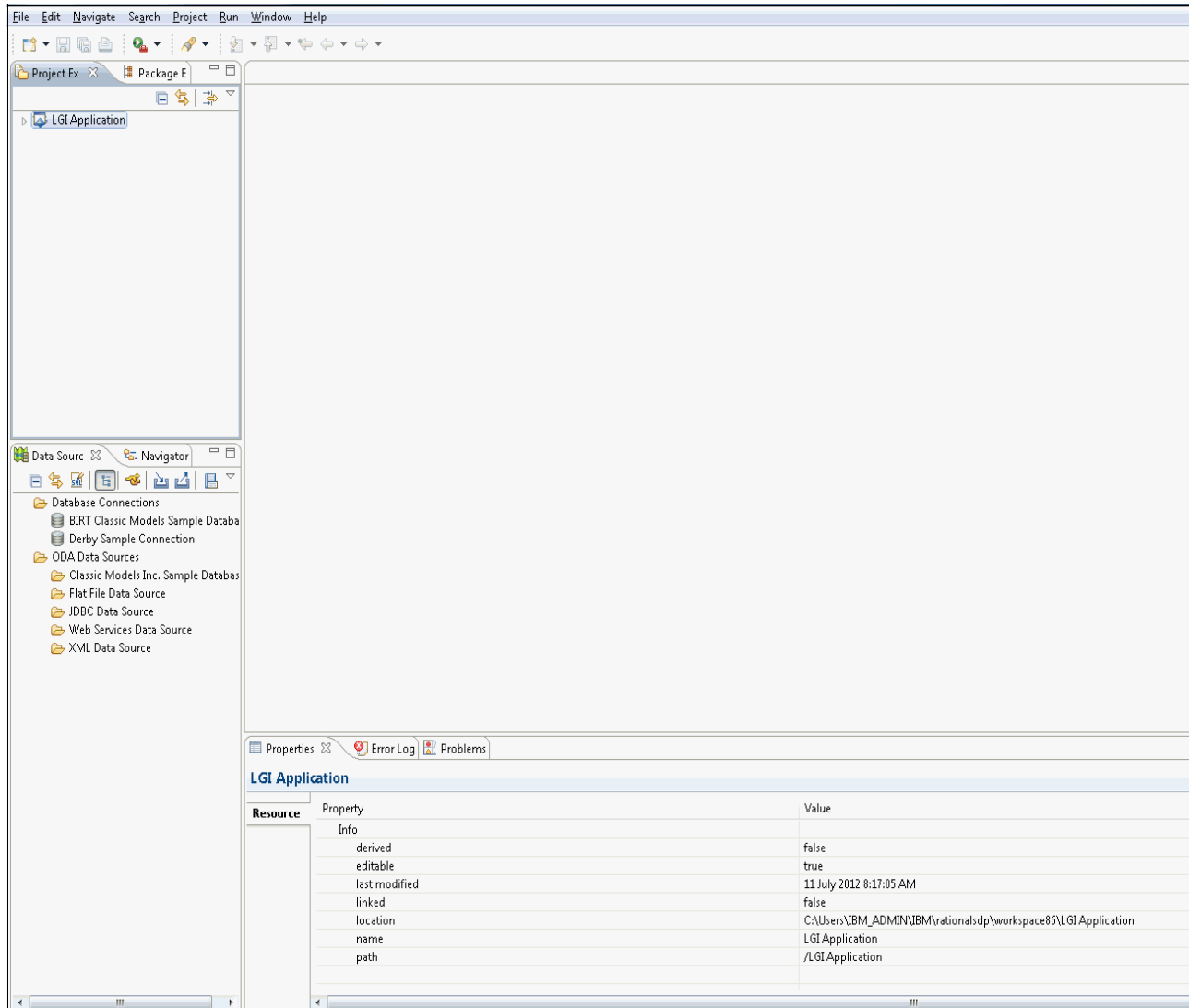


Figure 18 The LGI Application IMS Explorer Project

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

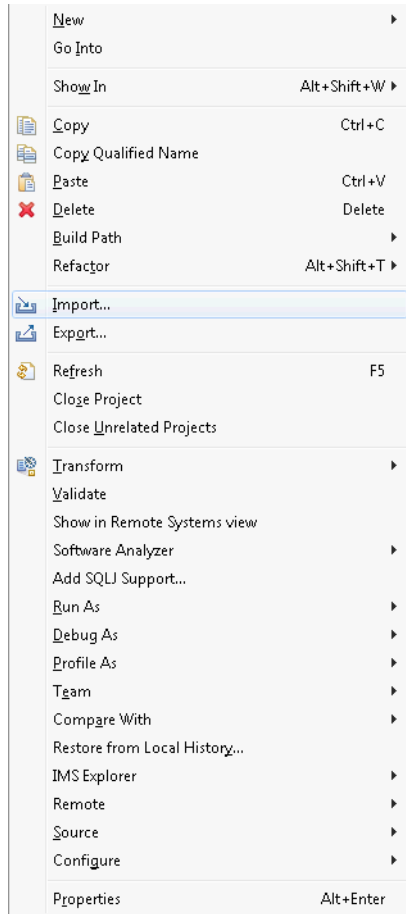


Figure 19 Options for an IMS Explorer Project

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

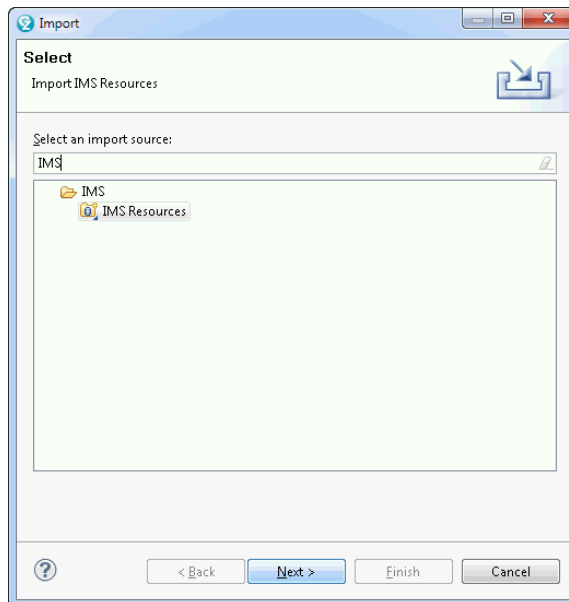


Figure 20 Select an Import Source

Enter the name of the Project in Figure 21.

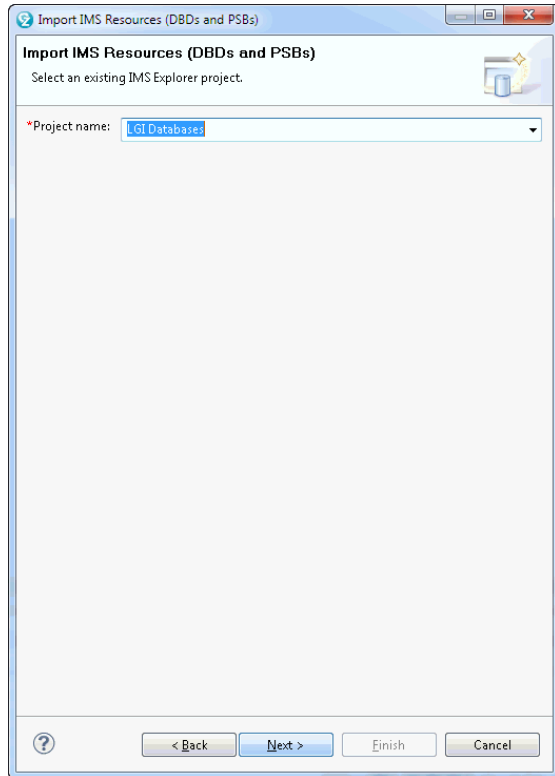


Figure 21 Provide a name for the IMS Resources to be imported

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

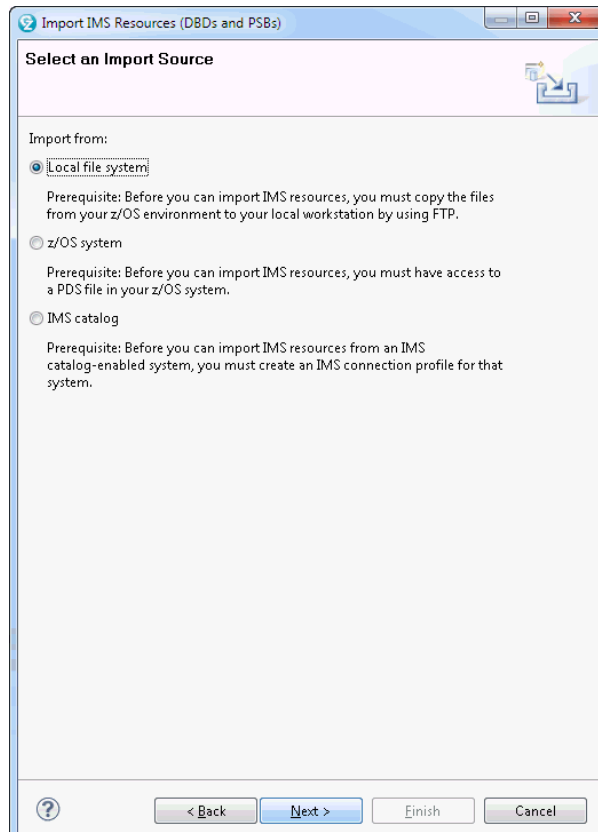


Figure 22 Select the Import Source of the source to be imported

You can now select the files from that source you want imported into your project. Note that while there are separate buttons on this window for DBDs and PSBs, IMS Explorer can identify the resources in the source file selected, and 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, so press the **Add DBD** button in Figure 23.



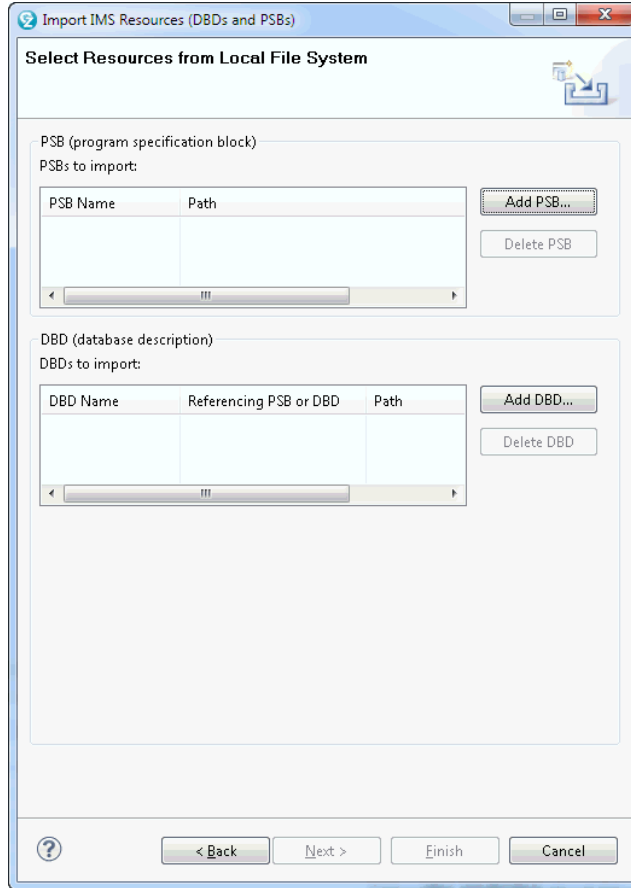


Figure 23 Select resources from the local file system

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

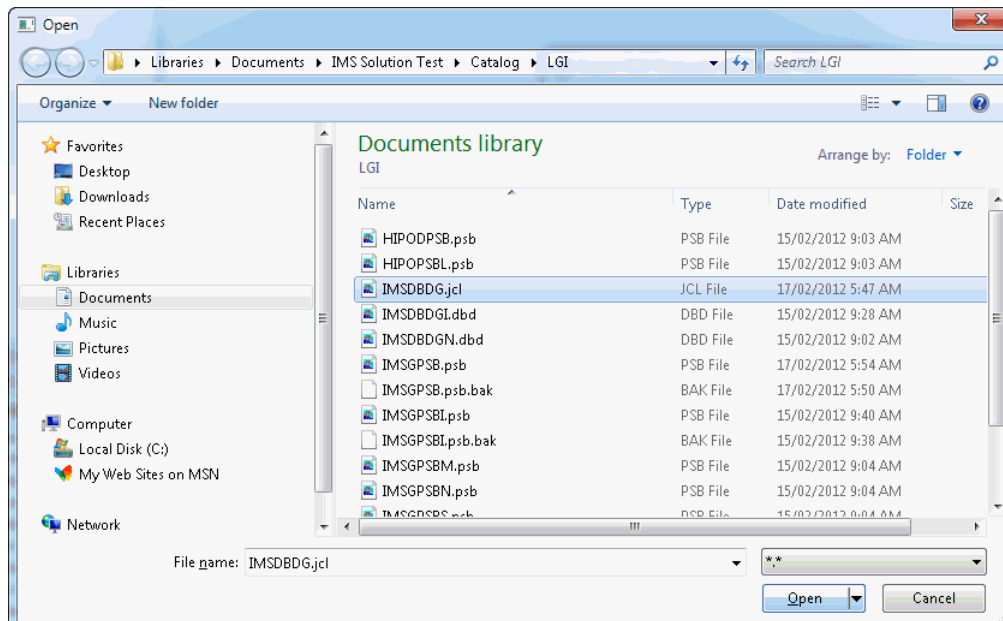


Figure 24 Select the file containing the resources to be imported

The Import IMS Resources window shows the two DBDs (S2U1DBD and S2U1DXD) which were in the single JCL file just imported. At this point, any of these DBDs can be deleted from the import list if desired.

To add PSBs to this import, press the **Add PSB...** button. See Figure 25.

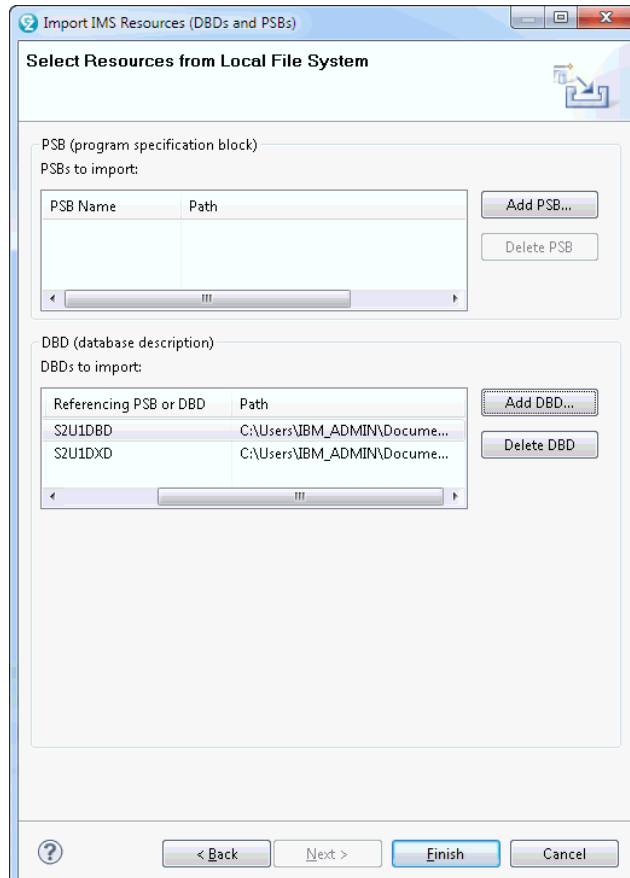


Figure 25 Keep selecting more files until you are done...

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 containing the PSBs you want to import, and press **Open**. See Figure 26.

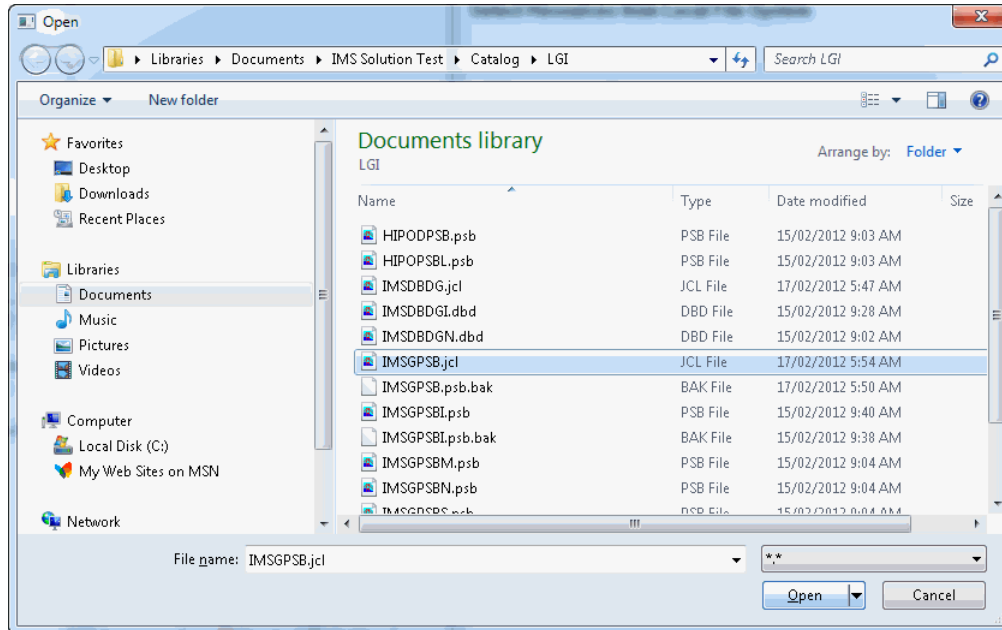


Figure 26 Selecting the PSB source to be imported

The Import IMS Resources window shows the two PSBs imported from the single file just selected. At this point, any of these DBDs and PSBs can be removed from the import list with the **Delete DBD** and **Delete PSB** buttons respectively.

Explorer also shows any DBDs that would 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.

Now, you are ready to import these DBDs and PSBs into your project. In Figure 27, press the **Finish** button.

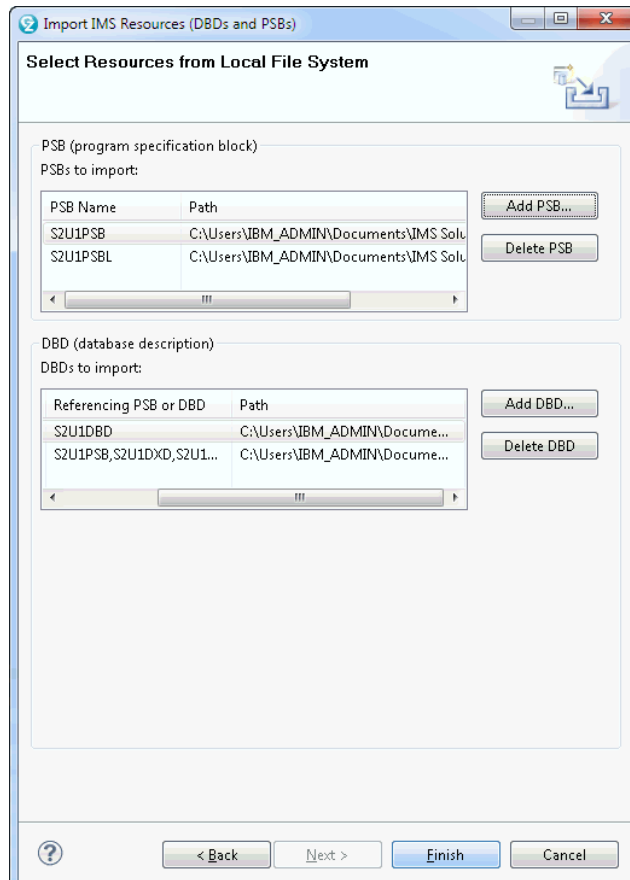


Figure 27 Lists of the DBD and PSB resources to be imported

The LGI Databases project now includes the DBDs and PSBs you have just imported, as well as several other artifacts needed later. At any time, you can refer back 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, since you have not yet added any of the metadata from the COBOL copybooks or PL/I Include members. You will do that soon.

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

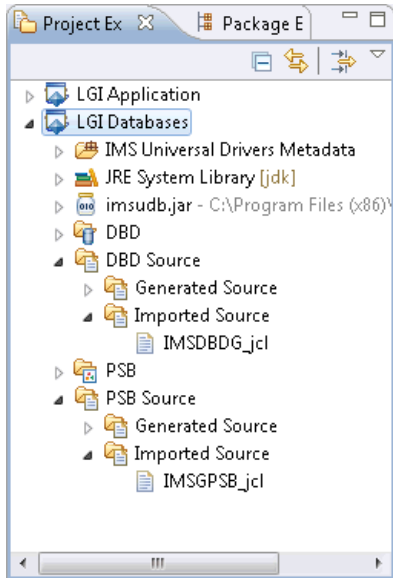


Figure 28 LGI Databases project, showing the DBD and PSB resources which have been imported

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

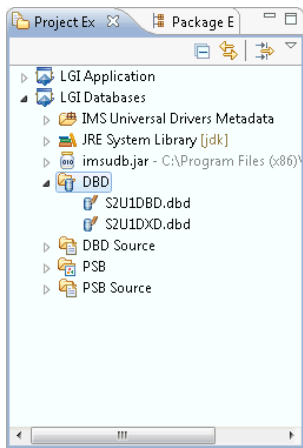


Figure 29 Expanding the DBD Resources view

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

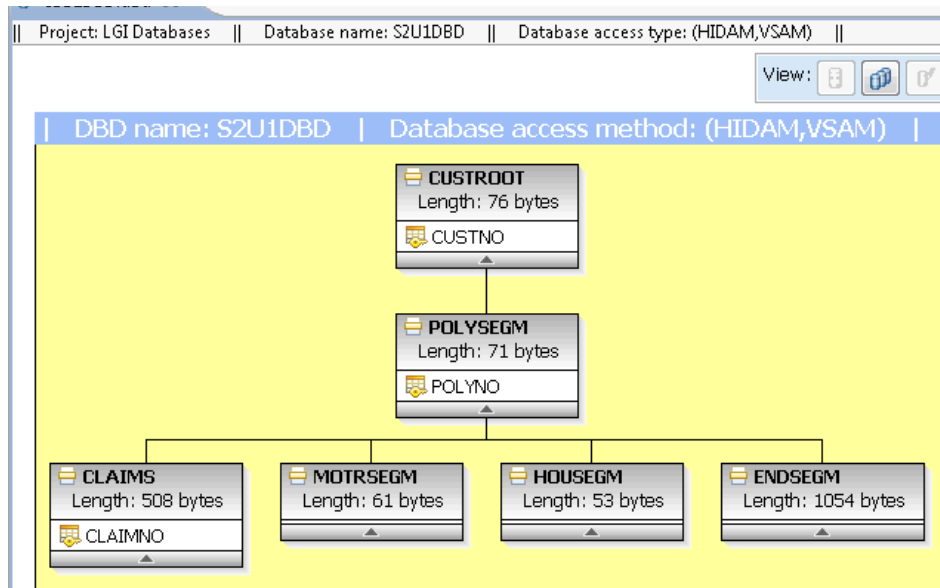


Figure 30 The Explorer view of an IMS Database

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

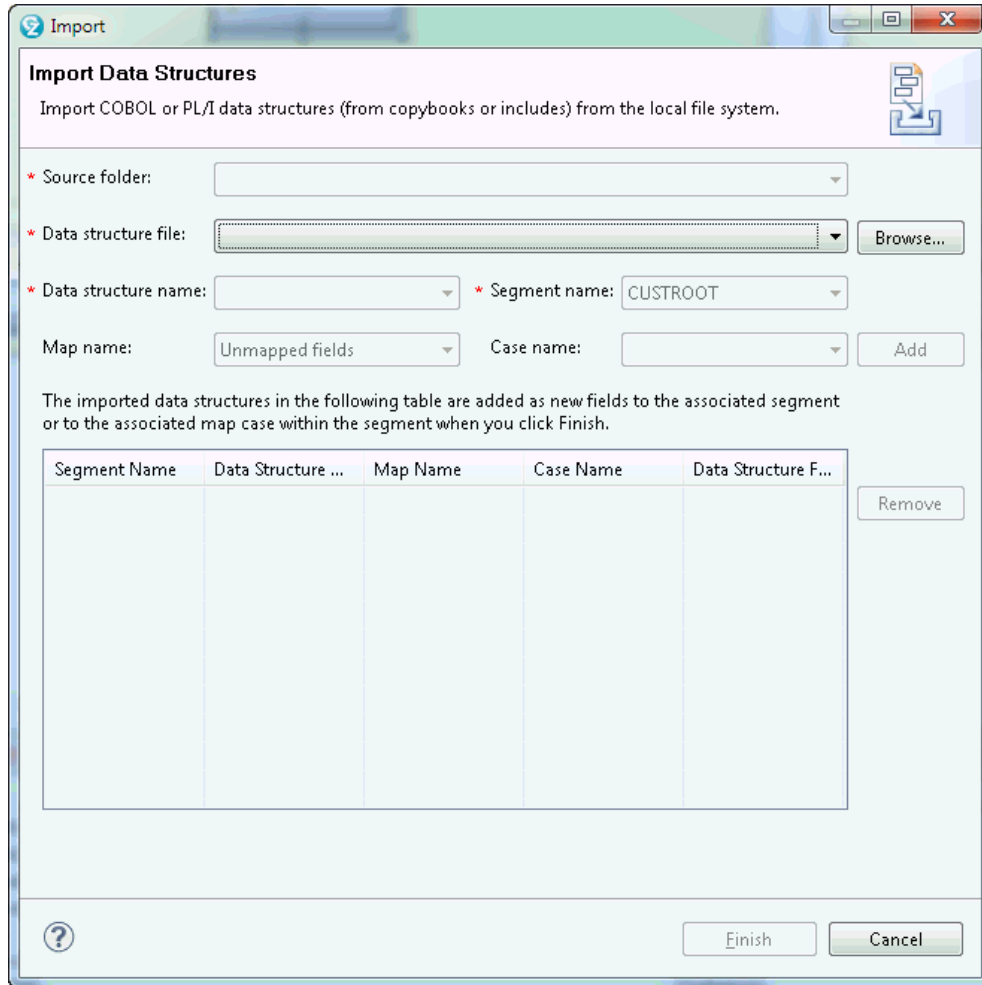


Figure 31 Importing database metadata from COBOL or PL/I

Click on **Browse** in Figure 32, to select the file containing the COBOL or PL/I definition for the segment.

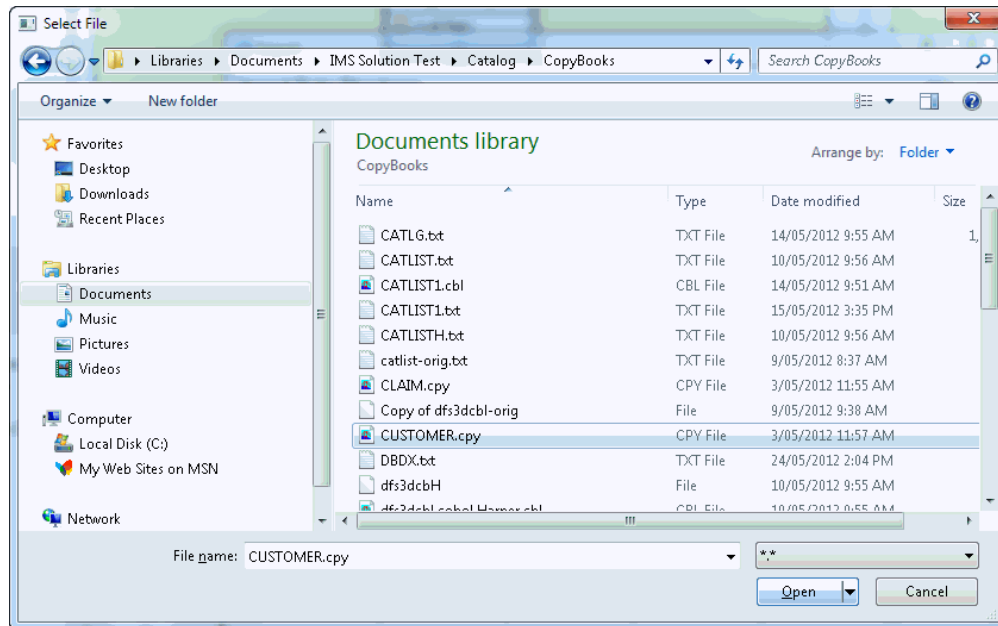


Figure 32 Select the file containing the Copybook or Include member

Select the file containing the COBOL or PL/I definition for the custroot (customer) segment, and press **Open** in Figure 33.

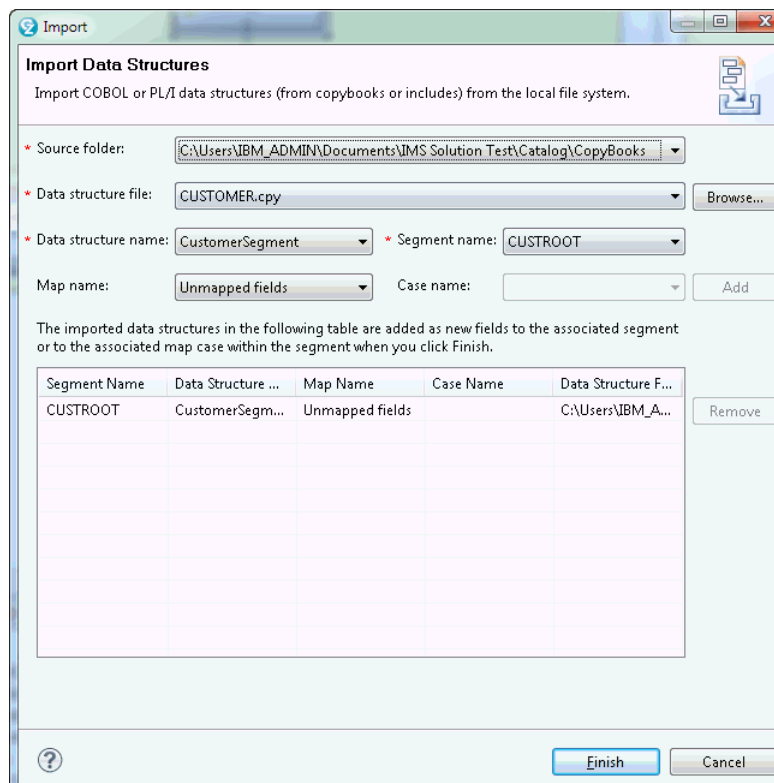


Figure 33 Identify the data structure to be imported for a database segment

In the COBOL copybook you selected, there is only one definition layout, so you can simply press the **Add** button Figure 34 to add this structure as the metadata for the CUSTROOT segment.



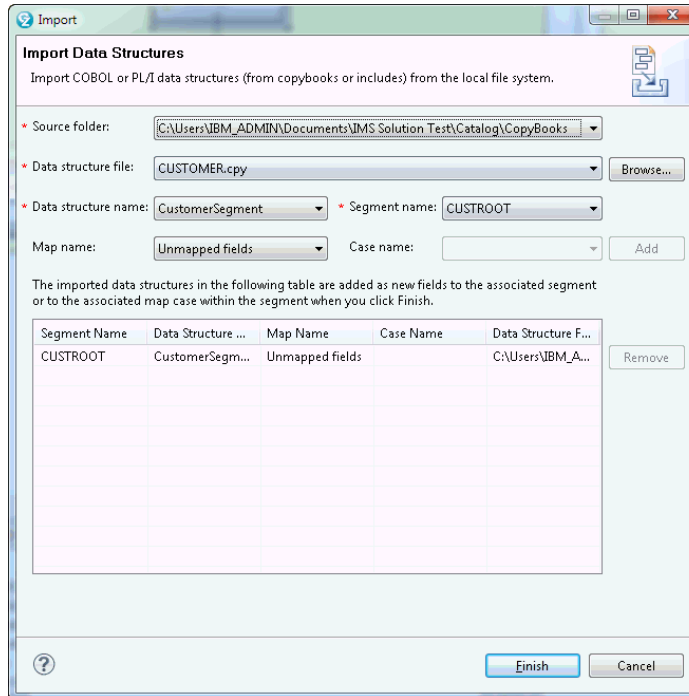


Figure 34 Importing the metadata for a database segment

From here, press the **Finish** button, 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.

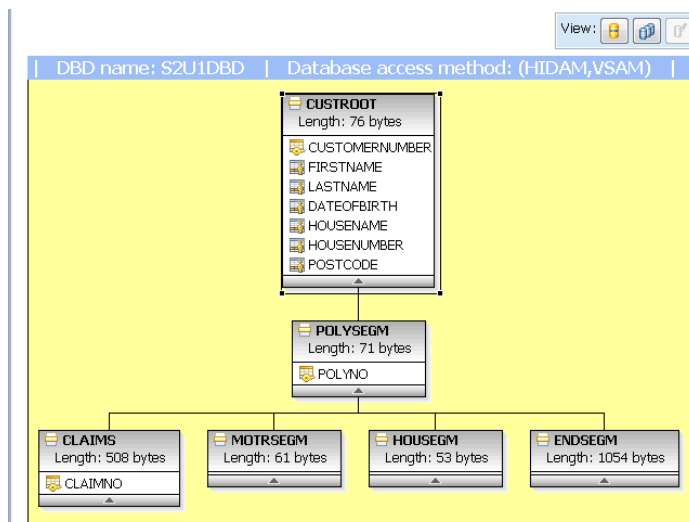


Figure 35 Explorer database view, with imported metadata for one segment

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

Figure 36 shows the database layout with definitions for each of the segments in the database. Note the asterisk at the left of the database name (\*S2U1DBD.dbd). This indicates the database definition has not yet been saved. Select **File -> Save** to save the enhanced database definition to the LGI Database Project.

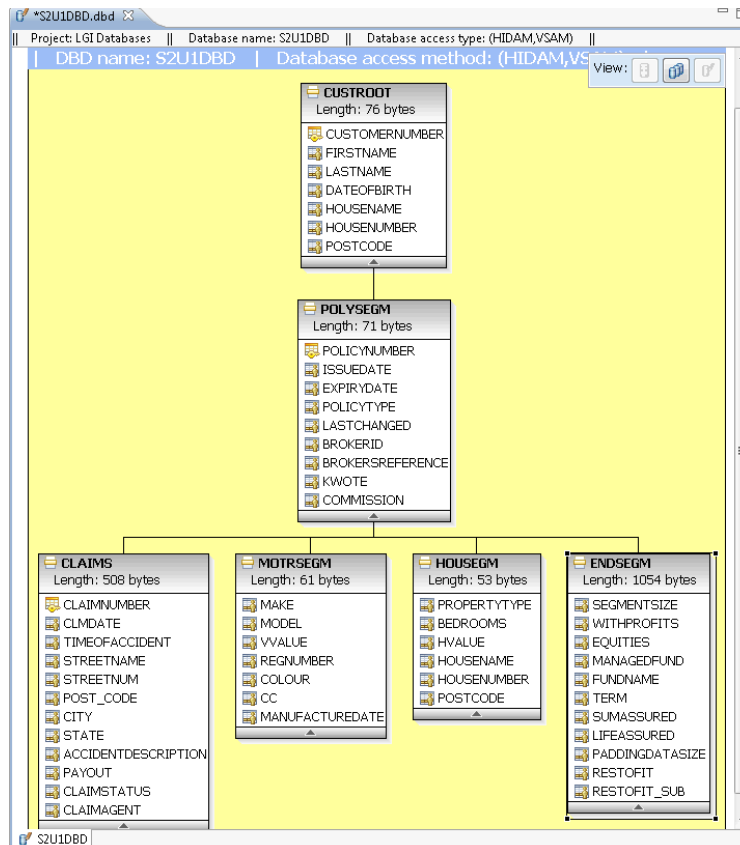


Figure 36 Explorer database view, with imported metadata for all segments

You have now captured the metadata for the layout of each of the segments in your database. If you click on the “twisties” for DBD Source and Generated Source in your project, you come to the “IMS catalog-enabled DBD source files” heading. Here are the enhanced definitions for the databases in your project. See Figure 37.

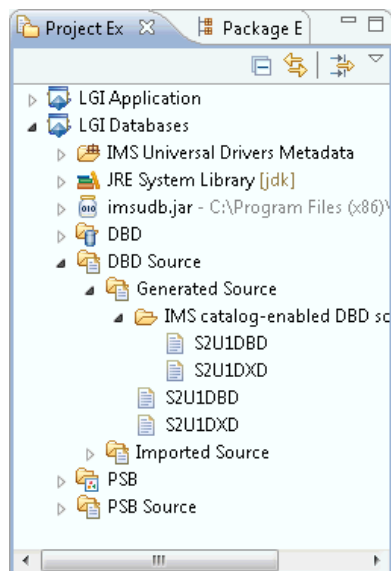


Figure 37 The catalog-enabled DBD source

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

```

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 steps include:

- ▶ Copy the expanded DBD source(s) to the host.
  - There are several ways to do this. Recent versions of IBM Personal Communications include an FTP client with a graphical user interface. Simple file transfer through your 3270 emulator is also available.
- ▶ 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 can be done in a single utility execution, or separately as desired.

The database metadata has now been captured into the catalog. Existing applications continue to access their databases as they always have, and 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 have being enhanced to leverage the IMS catalog as follows:

- ▶ 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

We describe the changes to the IMS drivers in two following sections:

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

### Access to the IMS databases from Java

Before any data can be accessed on DL/I databases, it is required to establish a connection. Traditionally an access with IMS/DLI requires a PSB, and this also true for Java. Traditional languages, like COBOL, PL/I... simply work with a segment content, which is accessed by the “Segment Search Arguments”, protocol. Java supports two ways to access DL/I databases through the universal drivers (Type 4 for distributed, Type 2 for local).

- ▶ SSA oriented (for DL/I clients)
- ▶ JDBC (Java Database Connectivity) clients

JDBC requires the additional support of metadata about 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 offer to the Java language a lot of interesting access perspectives (case mapping, structure selects...), not available in other languages.

The “metadata” support can come from the DLIModel or the IMS catalog, and the choice has to be made ad connection time, see Figure 39.

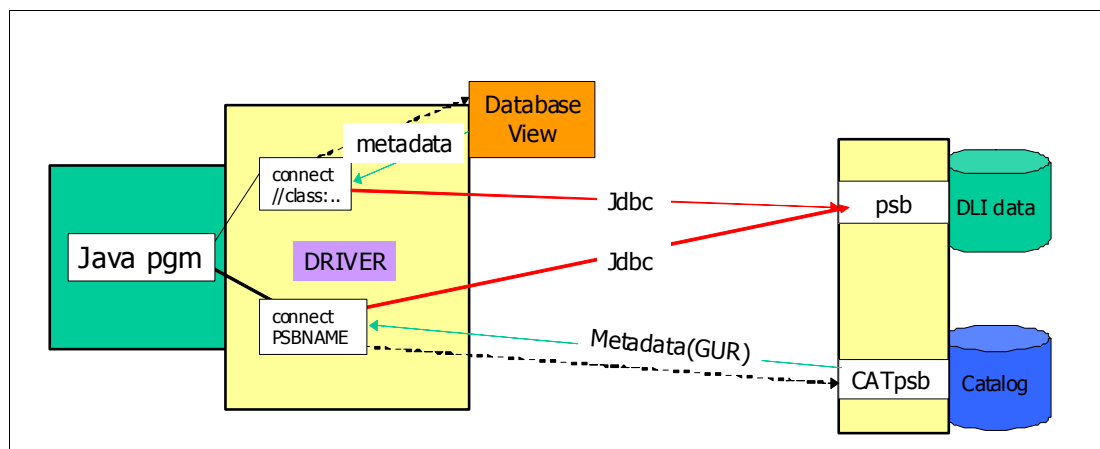


Figure 39 Connect of a Java program thru the drivers

Before being able to access the DL/I databases two elements are required

1. A PSB (for language Java) must exist

## 2. DatabaseView class or IMS catalog for obtaining metadata

- A DatabaseView class must have been generated by the DLIModel utility
- Metadata information is available in the catalog.

### Connecting to IMS for use with JDBC (JDBC client)

In Example 19 we show a code excerpt of how to obtain a Type 4 connection (with 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
    }
}
```

---

- Notice the use of the IMSDataSource, which gets its properties thru 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 then the database name (`url`) is the name of the PSB (program specification block) containing the database(s) to be accessed.
- If the metadata repository is the `DatabaseView` class then the database name is the fully qualified name of the Java metadata class generated by the `DLIModel` utility. In this case the name must begin with `class://`.

**Note:** 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

In Example 20 you can see an excerpt of the code required to obtain access to 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 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
    }

```

---

**Note:** 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...), reading a segment brings a complete record in an I/O area, basically this a byte area. This area has to be interpreted, as it 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 by overlaying the I/O area with dummy sections (COBOL copybooks, C, PL/I includes). This technique is not available for Java.

The changes on the DBDGEN macros, as explained in “New DBDGEN statements in IMS 12” and extensions to the Java “access” classes allow for an enhanced access approach of the DL/I databases by Java programs. They 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,DL/I) offers also a 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

**Note:** Traditionally, when creating a Struct or an Array object, you have to define it in a bottom up matter starting with the Struct attributes or the Array Elements.

IBM Extension allows for top down creation of a Struct or a 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.

Due to the *metadata* support for Java, several enhancements are now available for Java. We describe them 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 has been 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 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 two byte length (LL) field that identify the size of the segment instance. Universal Drivers are now sensitive to the LL field of a Variable Length Segment and manage the IO area of the segment instance on standard CRUD calls. A new Connection Property *llField* determines whether the ll field of the segment is exposed 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 exposed to the user in both the SQL and DL/I interfaces. User fully manages the LL field for the segment instance. This is how users currently manage variable length segments

► *llField* = false (default)

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

**Note:** The variable length support is introduced for both “DLIModel” and the “IMS catalog metadata”.

Example 21 shows the variable length information related to 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 stores a combination of values. Struct support was added specifically for the IMS catalog in IMS 12 to represent common application metadata.

Example 23 represents a simple structure with 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 shows the code for JDBC SQL retrieve of the structure (PERSON\_INFO), refer to COBOL definition in Example 10-20. In the code you extract the subfield values from the structure object, two approaches are shown.

*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 previous examples showed code for a JDBC client, Example 27 shows a Java code excerpt for a retrieve with 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 have been assembled in the catalog, arrays are also supported. Arrays are Data Structures that stores a repeating combination of values, see Example 28.

*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 only support Static Arrays, they can retrieve the metadata on Arrays as XML through the GUR call.

A proprietary method of setting the elements within a DBArrayElementSet object has been added, it 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(), 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 shows the retrieve of the segment "STUDENT", which contains an array. Notice the usage of the DBArrayElementSet class.

*Example 29 Retrieve of 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 {
    .....
}
```

---

```
//-----
```

Notice in the previous Example 29, the use of the DBArrayElementSet class.

Example 30 shows the insert of a new STUDENT segment, we show again the two ways of JDBC coding.

*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) are 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 runtime by the Universal drivers and the proper 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	MValue	Address	Make	Model	Year	HValue	Color
M	-	-	-	-	Ford	Escort	1989	2K	Red
H	Single Family	5	500K	555 Disk Drive Way, 95141	-	-	-	-	-

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

All case fields are exposed 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.

What you see in Figure 40 is the SQLview of the data. This representation gives the impression that a lot of space is empty on disk. The reality is different. All maps have to be of the same length and in this case the segment physical view would be like in Figure 41.

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

Note 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 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 interface and was designed to be functionally identical. The use of case-mapping supporting offers the following specific CRUD behaviors.

- ▶ SQL Select/DLI Read
 

When a case is not the active case based on the map's control field, its fields are treated as null fields
- ▶ SQL Insert/DLI Create
 

Cannot insert values for a case's fields unless the insert also includes the value for the control field that makes the case active
- ▶ SQL Update/DLI Update
 

Cannot update values for a case's fields unless the case is currently active or the control field is also being updated to a value that would make the case active
- ▶ SQL Delete/DLI Delete
 

No new behavior

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.getBytes("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 essentially overlapping fields. Redefines is a way of re-mapping 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).
```

---

In Example 34 you see 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>

```

---

Overlapping fields can be interchangeable. Search performance of a query depends on the type of field in the qualification statement

- ▶ Key Fields – Fastest
- ▶ Searchable Fields
- ▶ Not Searchable (i.e., application defined fields)

Universal drivers promote a field upwards when issuing queries against IMS

Example where the key field, KEY, and non-key field, NONKEY, redefine each other:

```

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

```

The trace and log file shows the promotion in the SSA list that is sent from the universal driver to IMS

## Public Converter Interfaces

Public interfaces have been added to the universal drivers internal type converters for customers to use in implementing their own type converter routines. Support has been added in IMS 10, IMS 11, and IMS 12

New ConverterFactory class that allows users to create:

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

Converter classes contains a getter and setter method for converting the data type to a binary representation

The IMS catalog is built to store information on fields with a user defined type, which allows users to interpret binary data stored in IMS in whatever form is required for their application

Example 35 show 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 input parameters are:

- ▶ `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 containing the metadata for the resources requested. The XML document conforms to IMS managed schemas. The following schemas are made available with the IMS catalog:



- ▶ DFS3XDBD.xsd
- ▶ DFS3XPSB.xsd

Example 37 shows the explicit GUR call.

*Example 37 Example of a explicit GUR call*

---

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

---

Example 38 shows the getCatalogMetaDataAsXML public method definition.

*Example 38 Example of 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 yyDDHHmmssff
 * @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

With a new function made available through the maintenance stream it is important to be current.

In this section, we summarize the recent maintenance for IMS 12 which provides and enhances the IMS catalog support. The list of APARs in Table 1 represents a snapshot of the current maintenance at the moment of writing. As such, the list becomes incomplete or even incorrect at the time of reading. Make sure 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, as well as to verify pre- and post-requisites.

Table 1 IMS 12 APARs for catalog support

APAR #	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

**Note:** PM42909 (UK78069) for IMS catalog support has changed the length (INQELEN in the DFSINQY mapping macro) of 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 using the INQY ENVIRON call.

The supporting documentation for the IMS catalog function is available online in the IMS Version 12 library in the Information Management for System z information center at Information Management Software for z/OS Solutions Information Center at <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 International Technical Support Organization, 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. Prior to his current role, Geoff was a member of the Worldwide IMS Advocate team for 12 years, providing Consulting, Education and Services to customers in many industries around the world. Geoff is the co-author of 12 IMS Redbooks® publications. Prior to 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 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 has specialized in Java, service-oriented architecture, IMS and WebSphere® applications, mainly on z/OS systems, and has 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 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 customer 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](http://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 August 28, 2012.

Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:  
[ibm.com/redbooks](http://ibm.com/redbooks)
- ▶ Send your comments in an email to:  
[redbooks@us.ibm.com](mailto: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

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®  
IBM®  
IMS™  
MVS™

Rational®  
Redbooks®  
Redbooks (logo) ®  
System z®

WebSphere®  
z/OS®  
zEnterprise®  
zSeries®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.