



Avoid Bind

Version 6.3

User's Guide



HLS Technologies, Inc.
3322 Sturbridge Lane
Sugar Land, Texas 77479-2223

Phone (281) 265-3004
Toll Free (888) 494-9019
Fax (281) 265-3006

hlstech@attglobal.net
<http://www.hlstechnologies.com/>



Avoid Bind User's Guide, Version 6 Release 3

Third Edition (June 2006)

This edition applies to Version 6 Release 3 of the Avoid Bind Product and to all subsequent releases until superseded by new editions of modified by technical documentation updates.

Reader comments on this document are welcomed and encouraged. Please send comments to:

HLS Technologies, Inc.
Technical Publications Group
3322 Sturbridge Lane
Sugar Land, Texas 77479-2223

© Copyright HLS Technologies, Inc. 2004. All rights reserved.

Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, or otherwise without the prior written consent of the publisher, HLS Technologies, Inc.



Table of Contents

Preface	iv
Who Should Use This Guide	iv
Prerequisites	iv
Summary of Changes	v
Introduction	1
DB2 and the Bind Process	1
The Precompiler.....	1
The DB2 Bind.....	2
What's the Problem?	3
Binds Create Procedural Bottlenecks.....	3
Binds Reduce Productivity	3
Binds Play "Access Path Roulette"	3
DB2 Version 2.3's Package Solution	4
The Cure for the DB2 Bind	4
Product Specifications	6
Special Features	6
Technical Description	7
Requirements	7
Technical Specifications	7
Installation and Operation	8
Installing Avoid Bind	8
Using Avoid Bind from DB2I	9
Using Special Features	10
DBRM Generation.....	10
DECLARE TABLE Exception	10
Coprocesor Support	10
Promote Verify	10
Promote Verify Parameters -	11
Coding of Parameters -	12
Appendices	14
Appendix A— JCL Samples	14
Appendix B - Messages and Codes	20
Notices	30
Trademarks	30



Preface

This user's guide describes how to install and use Avoid Bind. Avoid Bind is a software product that reduces the need to bind DB2 applications, thereby streamlining the development, testing, and implementation of DB2 applications as well as minimizing the effects of unexpected access path changes. Using Avoid Bind will reduce time and resources to increase developer productivity.

Who Should Use This Guide

This user's guide contains information on program uses, Installation, and procedures for Avoid Bind. The information is most useful to the following people:

- Individuals who need to determine applicability of Avoid Bind to their Installations
- System programmers who must install and support Avoid Bind
- Application programmers, system analysts, and database administrators who need to understand the role Avoid Bind plays in the application development process

Prerequisites

Most of the information contained herein is technical and specific to DB2, its associated programming languages, and Avoid Bind's application. The reader must have certain knowledge for this guide to be of use, such as the following:

- Experience in designing, developing, or implementing DB2 applications
- OR
- Experience managing people who perform these activities

For installing and/or supporting Avoid Bind you:

- Must have a basic familiarity with MVS utilities and JCL
- A working knowledge of SMP/E (strongly recommended)



Summary of Changes – Version 6 Release 3

Avoid Bind V6R3 contains the following changes and enhancements:

- Promote Verify – Compares the timestamp (consistency token) in the program being promoted to the timestamp currently recorded in the DB2 catalog.

Summary of Changes – Version 6 Release 2

Avoid Bind V6R2 contains the following changes and enhancements:

- DBRM Generation – Provides the option to recreate a missing DBRM based on information in the DB2 catalog.
- DECLARE TABLE exception – Provides the option to disregard changes in DECLARE TABLE statements when comparing old and new DBRMs.

Coprocessor support – Avoid Bind support for compilers that use the coprocessor (integrated precompiler).

Summary of Changes – Version 6 Release 1

Avoid Bind V6R1 contains the following changes and enhancements:

- Supports DB2 Version 8
- Minor technical enhancements
- Enhanced messages and diagnostics



Introduction

Avoid Bind is a software product that streamlines the development, testing, and implementation of DB2 applications by alleviating one of the principal bottlenecks affecting these processes: the DB2 bind. Why is it important to eliminate unnecessary DB2 binds?

As an application platform and enterprise data repository, DB2 continues to become more mission-critical with each passing day. Effective functioning of the DB2 platform has become critical to enterprise success. This means not only is the reliability of the online operation of the database necessary, but also the process of developing and implementing applications must become as efficient and free of roadblocks as possible. Eliminating roadblocks is exactly what Avoid Bind accomplishes.

Avoid Bind allows you to execute program modifications without the involvement of DB2 whenever possible. It permits you to safely bypass the DB2 bind process for source code changes that do not alter the existing SQL structure. On average, the use of Avoid Bind eliminates at least 50% of the bind activity in an installation. At some sites, the figure has exceeded 80%.

By eliminating unnecessary DB2 binds you can increase productivity and efficiency while reducing overhead. Avoid Bind makes your DB2 system run faster and makes an invaluable tool even better.

DB2 and the Bind Process

Preparing a DB2 application for execution involves numerous steps. The exact combination and sequence of steps depends upon the environment in which the application was developed and the environment in which it runs.

Nearly all applications require some form of compilation or language translation as well as linkage editing to create an executable load module. In addition, some applications require further preparation steps that are unique to their particular environments (e.g. a CICS newcopy).

The Precompiler

DB2's specific preparation steps include precompilation and binding, and the two are directly related. Before you can compile a program using imbedded (static) SQL, it must first be processed by the DB2 precompiler (DSNHPC).



The precompiler performs several essential functions:

- It performs extensive validation of the SQL statements.
 - a) Verifies syntax and structure.
 - b) Verifies matching host language variables and their attributes to the SQL statements that reference them.
- It replaces SQL statements with appropriate compliant host language statements (usually some form of subroutine call).
- It inserts a timestamp value into the source program.
- It extracts the SQL statements and stores them in condensed form as a Data Base Request Module (DBRM).

After the DB2 precompiler and the host language compiler successfully process a program with imbedded SQL, it must then undergo a DB2 bind before you can execute it.

The DB2 Bind

Binding establishes a fixed relationship between a program and the database(s) it will access. The bind process is complex and involves selecting access paths to the data, performing further SQL verification, and various other checks and validations. A program cannot access DB2 data until it has been successfully bound.

The DBRM the precompiler creates is the primary input to the bind process. DB2 uses information in the DBRM to create a valid connection between the application program and the DB2 database(s) it intends to access. The result of the bind is an application plan or package, which is stored to the DB2 Catalog.

The plan or package (the bind result) contains a timestamp value that, at execution time, is compared to the one found in the program. If the two timestamps do not agree, program execution fails with SQL return code -818 or -805. The timestamp validation process is DB2's way of forcing you to rebind the applications after every compilation, whether or not you modified the SQL statements.



What's the Problem?

Though binding is necessary for the DB2 system to run properly, unnecessary or extraneous binding is harmful to the system. It acts as a procedural bottleneck, lowers productivity, and forces you into an involuntary game of “access path roulette.” This section will expand on each of these problems dealing with irrelevant DB2 binds.

Binds Create Procedural Bottlenecks

The bind process can be an expensive bottleneck, and should be avoided whenever possible! The need to bind applications before using them is costly in many ways.

In most cases, DB2 must completely unbind access to a plan in order to replace it. Recreating these new access points is exactly what rebinding does, and in busy online transaction processing environments, the time consuming act of rebinding to update the program may be impossible to do during normal business operations. The end result is that the company can't move the newly updated applications into production when needed.

Binds Reduce Productivity

Programmer productivity obviously suffers when they must wait hours, or even overnight, to transfer updated applications to test or execute. Also, if the modified application is not available to the end user when needed, you lose their productivity as well. Even worse, you could lose more business by temporarily shutting down an entire application or online system in order to implement an emergency change because of the rebinding process.

Binds Play “Access Path Roulette”

Every time you perform a bind you play an involuntary game of “access path roulette.” In short, whenever DB2 binds (or rebinds) a program, it reevaluates the access paths used to read or write the data. Often, this reevaluation ends up selecting new or different access paths, “access path roulette.” However, DB2 issues no warning message to alert you when this happens! Usually, your first indication that the access paths have changed comes when an application's response time suddenly changes from good to terrible for no apparent reason. This means the rebinding process can slow your system *and* never warn you it's going to happen!



DB2 Version 2.3's Package Solution

IBM recognized the inherent problem with binding, especially with large multi-DBRM plans, and attempted to mitigate it with DB2 version 2.3. This release introduced a new type of plan called a package, and while the use of packages *can* reduce the size and resource consumption of some binds; it does not eliminate the need to do them.

Although using packages can make the bind process a bit less disruptive, it still requires DB2 to acquire various locks to move forward and this can create serious contention problems that may impact throughout, just as in the pre-package days. Packages do very little to relieve the basic bottleneck of unnecessary binding and do nothing to address the problem of “access path roulette.” Only Avoid Bind can eliminate all these problems and truly increase efficiency and reduce overhead.

The Cure for the DB2 Bind

However you look at it, unnecessary binds are expensive and counterproductive. Avoid Bind is the answer to eliminate these troublesome blocks to efficiency and productivity.

The cure for unnecessary bind overhead is Avoid Bind. Avoid Bind automatically analyzes each Precompile to determine whether the SQL structure has changed; In other words, whether it really needs the bind or not. When you use Avoid Bind the DB2 system will only create required binds, and you'll never have to perform an extraneous bind and waste your time again.

Avoid Bind also pays dividends in the area of change management. When integrated into a production promotion scheme, Avoid Bind automatically detects those production application changes requiring a bind. DBA's can then concentrate their efforts on just those changes that will affect the SQL structure.

Avoid Bind:

- Analyzes each Precompile to determine if the SQL structure has changed
- Determines if the bind is necessary
- Allows you to circumvent the bind process whenever possible
- Automatically detects if the change made requires a bind and allows DBA's to concentrate on the necessary changes that affect the SQL



Using Avoid Bind will allow you to:

- Increase speed
- Reduce overhead costs
- Concentrate on necessary changes and avoid extra work
- Strengthen efficiency



Product Specifications

Avoid Bind monitors the execution of the DB2 precompiler and, by analyzing both the new and old DBRMs, determines whether or not a bind is required. The process must have access to the library containing the DBRMs that were used during the previous bind operation. It assumes that the information in this DBRM library is identical to that used by DB2. Avoid Bind does *not* connect to an active DB2 subsystem to validate DBRM and plan information, *except* when the optional DBRM Generation feature is used. The following section, Special Features, describes the DBRM Generation feature in more detail.

Special Features

Avoid Bind includes three special features: DBRM Generation, the DECLARE TABLE exception, and coprocessor support. These special features enhance the abilities of Avoid Bind, and make using the software easier.

DBRM Generation – Avoid Bind works by comparing the results of the current precompile (the “new” DBRM) to the results of the previous precompile (the “old” DBRM), so it is usually necessary to have the old DBRM available when you execute Avoid Bind. However, some Installations do not save DBRMs as a matter of policy, and in these cases the old DBRM is not available for Avoid Bind to compare against.

The *DBRM Generation* feature circumvents this limitation by generating a temporary working copy of the old DBRM from the information in the DB2 catalog. The comparison can then proceed normally, and it deletes the temporary DBRM when the job ends.

Note: You must use JCL to invoke *DBRM Generation* and/or *DECLARE TABLE exception*. For detailed information see “Using Special Features” on page 9.

DECLARE TABLE exception – When activated, *DECLARE TABLE exception* causes Avoid Bind to ignore DECLARE TABLE statements when comparing the old and new DBRMs. Like DBRM Generation, you invoke this feature through JCL. For detailed information see “Using Special Features” on page 9.

Coprocessor Support – In the traditional scenario, the DB2 precompiler runs as a separate job step. In this environment, Avoid Bind intercepts the source code after the precompiler modifies it, and before the language



compiler sees it. Avoid Bind operates on this “intermediate” source code before passing it on to the compiler.

In the coprocessor environment the precompiler and the language compiler are combined into a single step, so the intermediate source code is not available for manipulation. With Avoid Bind *Coprocessor Support*, Avoid Bind functions in the coprocessor environment so that bind avoidance continues to function.

Promote Verify – When promoting applications to production, *Promote Verify* compares the timestamp in the program being promoted to the timestamp currently recorded in the DB2 catalog. If they agree, no bind is required. The program sets a return code to indicate whether or not this is the case. *Promote Verify* can process multiple applications in a single step.

Technical Description

The following section contains the requirements and technical specifications for Avoid Bind.

Requirements

- Access to the DBRM used by the most recent bind (except when using DBRM Generation)
- Any of the following programming languages: VS COBOL, COBOL-II, PL/I, or Assembler

Technical Specifications

- Avoid Bind does not require APF authorization, nor does it involve the use of “hooks” or “back doors” into DB2 or MVS.
- Avoid Bind is available by Internet download or email. The product will be delivered as a zipped uploadable job stream, and Installation is described in detail in “Installation and Operation” page 8.
- Avoid Bind consists of a single load module and is CPU serial number sensitive; Avoid Bind requires a password to permit execution.
- Avoid Bind supports all versions and releases of DB2.



Installation and Operation

This section contains information on how to install and operate Avoid Bind and its special features. Specific examples of the operation of the program are available in the appendices, as well as tables listing messages and return codes.

Installing Avoid Bind

Avoid Bind comes as an uploadable *.ZIP* file. The *.ZIP* file contains two files: The user's guide (this document) and the install file. To install Avoid Bind

1. Upload the install file to your host. See note below.

Note: Be sure to use a binary file transfer and upload the file to a preallocated MVS data set with the following attributes: **RECFM=FB**, **LRECL=80**, and **DSORG=PS**. Block size may be any valid multiple of 80.

2. Once the install file has been uploaded to your MVS host, it must be unpacked, *i.e.*, converted back to its original format (partitioned data set). Use the TSO *Receive* command to do this. The format of this command is

```
TSO RECEIVE INDA( 'data.set.name' )
```


3. In the command format shown above, *data.set.name* would be replaced by the actual data set name you assigned to the uploaded file. For example, if the file were named USER55.AVB630.UPLOAD, then you would issue the command shown below:

```
TSO RECEIVE INDA( 'USER55.AVB630.UPLOAD' )
```

4. When you issue the *Receive* command, you will be prompted with message INMR906A. At this time you can just press ENTER, which will cause the reconstituted PDS library to be created using your TSO ID as the high level qualifier, or you can enter the subcommand

```
DA( 'data.set.name' )
```

...which will cause the PDS to be generated using the name you specify.



5. Once the *Receive* command has been executed successfully, you should have a PDS library containing the elements needed to install Avoid Bind. To install the product, follow the instructions in the README member and the INSTALL jobs.

Using Avoid Bind from DB2I

To invoke Avoid Bind from a foreground DB2 application, use the following procedure.

1. Locate the CLIST named DSNH. The default SLIST library for DB2I is named DSN vrm .SDSNCLST, where vrm is the version, release, and modification level of DB2. (*Example: DSN710.SDSNCLST*)
2. Within the CLIST, find the statement that initializes the variable PCLOAD. If your CLIST is the default version as shipped by IBM, the definition should look something like this:

```
PCLOAD(''DSN710.SDSNLOAD(DSNHPC)'') +
```

3. Make one of the following changes, depending on whether or not the DB2I load library is in the MVS link list (LINKLIST).
 - a) If the DB@I load library *is* in LINKLIST, modify the definition in the CLIST so that it looks like this:

```
PCLOAD(''youravb.loadlib($AVB)'')
```

- b) If the DB2I load library is *not* in LINKLIST, then you must copy and \$AVB load module to the DB2I load library and modify the definition in the CLIST so that it looks like this:

```
PCLOAD(''DSN $vrm$ .SDSNLOAD($AVB)'')
```



Using Special Features

This section contains some specific information concerning the use of the special features associated with Avoid Bind. These special features include DBRM Generation, DECLARE TABLE exception, and Coprocessor support.

DBRM Generation

You activate DBRM Generation by adding a special DD statement to the execution JCL. The DDname must begin with the characters “SSID”; this signals Avoid Bind that DBRM Generation is in effect. The remainder of the DDname (one to four characters) specifies the subsystem name of the DB2 subsystem from which it will extract the DBRM information. The following example uses subsystem DSN1 to invoke DBRM Generation.

```
//SSIDDSN1 DD DUMMY
```

DECLARE TABLE Exception

As with DBRM Generation, you activate the DECLARE TABLE Exception by adding a special DD statement to the execution JCL. The DDname of the statement must be “SKIPDECL”. Please take a look at the following example.


```
//SKIPDECL DD DUMMY
```

Coprocessor Support

Coprocessor Support requires completely different JCL than traditional Avoid Bind. It has a different program name, as well as different DD statements. Coprocessor Support adds two additional job steps; one before the compiler step and another after. Please take a look at the example presented in Appendix A on page 14.

Promote Verify

The Promote Verify function is designed to be used just before promoting an application to production. The function compares the timestamp in the program being promoted to the timestamp currently recorded in the DB2 catalog. If they agree, no bind is needed. This lets you promote with greater



confidence that the application will run without error. Promote Verify is comprised of 2 programs, AVB2 and AVB3. AVB2 processes a single application with each execution; AVB3 can process multiple applications in a single job step.

AVB2

AVB2 is executed in batch, via JCL. In the JCL, ddname DBRMIN points to the DBRM of the program to be promoted. The control parameters entered via SYSIN determine which catalog entry to compare the DBRM to. You use the control parameters to specify either a DBRM within a plan, or a package within a collection. Promote Verify compares the timestamp in the DBRM to the timestamp in the specified catalog entry and issues the appropriate return code.

AVB3

AVB3 is executed in batch, via JCL. The function and purpose of AVB3 are similar to those of AVB2; however, AVB3 is designed to process multiple applications in a single step. AVB3 reads a set of bind commands from ddname BINDIN. For each bind command, Promote Verify searches the library referenced by ddname DBRMLIB to locate the corresponding DBRM. The consistency token is extracted from the DBRM and saved.

AVB3 checks the DB2 subsystem catalog for the corresponding DBRM or package entry, as indicated in the bind command. The consistency token in the catalog entry is compared to the one saved from the DBRM. If they are equal, the bind is unnecessary and the “rejected” bind command is written to ddname BINDREJ. If the tokens are unequal, a bind is required; in this case the command is written to ddname BINDOUT.

When AVB3 terminates, the commands written to BINDOUT may be executed by a subsequent job step. In this way only necessary binds are performed; unnecessary binds are skipped.

Promote Verify Parameters –

Note: These parameters apply *only* to AVB2.

Control parameters must be entered one per line. Any line beginning with an asterisk (*) is a comment and is ignored by the program. There are five control parameters. They may be entered in any order, but only certain combinations are valid. The five parameters are:

SSID
DBRM
PLAN



PACKAGE COLLID

Only certain combinations of parameters are valid. The valid combinations are:

SSID, DBRM, and PLAN
or
SSID, PACKAGE, and COLLID

Any other combination is treated as an error.

Coding of Parameters –

Note: These parameters apply *only* to AVB2.

SSID

The SSID parameter identifies the DB2 subsystem to be checked. It must be specified as a one-to-four-character subsystem name. Example:

SSID=DB7G

DBRM

The DBRM parameter specifies the DBRM to be checked in the DB2 catalog. It is used along with PLAN to identify the catalog entry whose timestamp is to be compared to the DBRM timestamp. Example:

PLAN=ABC77

PLAN

The PLAN parameter specifies the plan that contains the DBRM. It is used along with DBRM to identify the catalog entry whose timestamp is to be compared to the DBRM timestamp. Example:

PLAN=PRL500

PACKAGE

The PACKAGE parameter specifies the package to be checked in the DB2 catalog. It is used along with COLLID to identify the catalog entry whose timestamp is to be compared to the DBRM timestamp. Example:

PACKAGE=Z25UPDT



COLLID

The COLLID parameter specifies the collection that contains the package. It is used along with PACKAGE to identify the catalog entry whose timestamp is to be compared to the DBRM timestamp. Example:

```
COLLID=ACUPDT
```



Appendices

Appendix A contains JCL samples for the different methods of using Avoid Bind with and without the special features. There are examples of Traditional Avoid Bind, Avoid Bind with DBRM Generation, Avoid Bind with DECLARE TABLE Exception, and the Coprocessor Support.

Appendix B contains the messages and return codes for Avoid Bind. You can reference this section when you need the specific meaning of a message from the program.

Appendix A— JCL Samples

Note: In the JCL Samples, strings surrounded by question marks (?) represent parameters you need to replace with values appropriate for your Installation.

Example 1 – Traditional Avoid Bind

This example illustrates a typical Avoid Bind job step not using any special features. The Avoid Bind step takes the place of a normal precompiler step; it is normally followed by a compiler step that specifies the &&COMPIN data set as its SYSIN.


```
//*  
/*****  
/* Precompile the Source *  
/*****  
/*  
//PC EXEC PGM=$AVB,PARM='HOST(COB2),APOST'  
//STEPLIB DD DISP=SHR,DSN=?avb.loadlib?  
// DD DISP=SHR,DSN=?db2.loadlib? (if not in LNKLST)  
//SYSLIB DD DISP=SHR,DSN=?db2.precompiler.syslib?  
//DBRMLIB DD DISP=SHR,DSN=?existing.dbrmlib(dbrmname)?  
//SYSCIN DD DISP=(,PASS),DSN=&&COMPIN,UNIT=SYSALLDA,  
// SPACE=(CYL,(5,2))  
//SYSUT1 DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA  
//SYSUT2 DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA  
//AVBWK1 DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA  
//SYSPRINT DD SYSOUT=*  
//SYSTEM DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
/*
```



Example 2 – Avoid Bind with DBRM Generation

This example illustrates the DBRM Generation feature. The JCL is identical to Example 1 with the addition of the SSID DD statement. This statement requests DBRM Generation and specifies PRO3 as the DB2 subsystem whose catalog is to be used for deriving the DBRM information.

```
//*
/*****
/* Precompile the Source *
/*****
/*
//PC EXEC PGM=$AVB,PARM='HOST(COB2),APOST'
//STEPLIB DD DISP=SHR,DSN=?avb.loadlib?
// DD DISP=SHR,DSN=?db2.loadlib? (if not in LNKLST)
//SSIDPRO3 DD DUMMY
//SYSLIB DD DISP=SHR,DSN=?db2.precompiler.syslib?
//DBRMLIB DD DISP=SHR,DSN=?existing.dbrmlib(dbrmname)?
//SYSCIN DD DISP=(,PASS),DSN=&&COMPIN,UNIT=SYSALLDA,
// SPACE=(CYL,(5,2))
//SYSUT1 DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA
//SYSUT2 DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA
//AVBWK1 DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
/*
```



Example 3 – Avoid Bind with DECLARE TABLE Exception

This example illustrates the DECLARE TABLE Exception. The JCL is identical to Example 1 with the addition of the SKIPDECL DD statement. This statement requests Avoid Bind to disregard DECLARE TABLE statements when comparing the old and new DBRMs.

```
//*
/*****
/* Precompile the Source *
/*****
/*
//PC      EXEC   PGM=$AVB,PARM='HOST(COB2),APOST'
//STEPLIB DD DISP=SHR,DSN=?avb.loadlib?
//        DD DISP=SHR,DSN=?db2.loadlib?      (if not in LNKLST)
//SKIPDECL DD DUMMY
//SYSLIB  DD DISP=SHR,DSN=?db2.precompiler.syslib?
//DBRMLIB DD DISP=SHR,DSN=?existing.dbrmlib(dbrmname)?
//SYSCIN  DD DISP=(,PASS),DSN=&&COMPIN,UNIT=SYSALLDA,
//        SPACE=(CYL,(5,2))
//SYSUT1  DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA
//SYSUT2  DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA
//AVBWK1  DD SPACE=(CYL,(4,2)),UNIT=SYSALLDA
//SYSPRINT DD SYSOUT=*
//SYSTEM  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
/*
```

Example 4 – Coprocessor Support

This example illustrates the use of Avoid Bind with the integrated coprocessor. The coprocessor integrates the DB2 precompiler with the language compilers (COBOL in our example). Using Avoid Bind in this environment requires that you add two extra job steps to the compile JCL. The following example illustrates a COBOL compilation with the coprocessor and Avoid Bind.

```
//*
/*****
/*      Save a working copy of the old DBRM      *
/*****
/*
//SAVE      EXEC   PGM=IEBGGENER
//SYSPRINT DD   SYSOUT=*
//SYSIN     DD   DUMMY
//SYSUT1   DD   DISP=SHR,DSN=?existing.dbrmlib(dbrmname)?
//SYSUT2   DD   UNIT=SYSALLDA,SPACE=(CYL,(1,1)),
//          LRECL=80,RECFM=FB,
//          DISP=(,PASS),DSN=&&OLDDBRM
/*
/*****
/*      Compile the target program      *
/*****
/*
//COB      EXEC   PGM=IGYCRCTL,REGION=4M,
//          PARM=' SQL(HOST(ASM),STDSQL(NO)) '
//STEPLIB DD   DISP=SHR,DSN=IGY320.SIGYCOMP
//SYSLIB  DD   DISP=SHR,DSN=SYS1.MACLIB
//          DD   DISP=SHR,DSN=SYS1.MODGEN
//          DD   DISP=SHR,DSN=DSN710.SDSNMACS
//SYSPRINT DD   SYSOUT=*
//SYSTEM  DD   SYSOUT=*
//SYSUT1  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSUT2  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSUT3  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSUT4  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSUT5  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSUT6  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSUT7  DD   UNIT=SYSALLDA,SPACE=(CYL,10)
//SYSLIN  DD   UNIT=SYSALLDA,SPACE=(CYL,(1,1)),
//          LRECL=80,RECFM=FB,
//          DISP=(,PASS),DSN=&&OBJECT
//DBRMLIB DD   DISP=OLD,DSN=?dbrmlib.data.set.name(member)?
//SYSIN   DD   DISP=(OLD,DELETE),DSN=&&SYSCIN
/*
/*****
/*      Execute Avoid Bind      *
/*****
/*
//AVB1     EXEC   PGM=AVB1
//STEPLIB DD   DISP=SHR,DSN=?avb.loadlib?
//SYSPRINT DD   SYSOUT=*
//TDUMP    DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
//OLDDBRM DD   DISP=(OLD,DELETE),DSN=&&OLDDBRM
//NEWDBRM DD   DISP=OLD,DSN=?dbrmlib.data.set.name(member)?
//OBJECT   DD   DISP=(OLD,PASS),DSN=&&OBJECT
//SYSUT1   DD   DUMMY
```



Example 5 – Promote Verify

This example illustrates the use of the Promote Verify feature. In the JCL, ddname DBRMIN points to the DBRM of the program to be promoted. The control parameters entered via SYSIN determine which catalog entry to compare the DBRM to. You use the control parameters to specify either a DBRM within a plan, or a package within a collection. (See pg 10 for information on coding the parameters.) Promote Verify compares the timestamp from the DBRM to the timestamp in the specified catalog entry and issues the appropriate return code.

```
//AVB2 EXEC PGM=AVB2
//STEPLIB DD DISP=SHR,DSN=AVB.LOADLIB
//          DD DISP=SHR,DSN=DSN710.SDSNLOAD
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DBRMIN   DD DISP=SHR,DSN=DEV.REL.DBRM(HLSA024)
//SYSIN    DD *
*
* This is a comment
*
SSID=DB8G
PACKAGE=HLSA024
COLLID=HLS24C
/*
```



Example 6 – Promote Verify with Multiple applications

You invoke Promote Verify using JCL. Sample JCL for an AVB3 job step is shown below. Following the example is a brief discussion of the ddnames required and the purpose of each.

```
//STEP1 EXEC PGM=AVB3
//STEPLIB DD DISP=SHR,DSN=AVB3.LOADLIB
// DD DSN=DSN810.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSEXPLN DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//BINDOUT DD DSN=&&BINDCMDS,DISP=(NEW,PASS),
// UNIT=SYSALLDA,SPACE=(TRK,15,RLSE),
// RECFM=FB,LRECL=80
//BINDREJ DD SYSOUT=*
//DBRMLIB DD DISP=SHR,DSN=USER.DBRMLIB
//BINDIN DD DISP=SHR,DSN=USER.BIND.COMMANDS
```

- STEPLIB** Identifies the load library containing the Promote Verify program.
- In the example, this ddname also references the DB2 load library for the subsystem in which the binds are to be performed. If the DB2 load library is in link list – as it often is – it may be omitted from STEPLIB.
- SYSPRINT** Defines a sequential data set used for listing control statements and messages.
- SYSEXPLN** Defines a sequential data set used for listing messages.
- SYSOUT** Defines a sequential data set used for listing messages
- BINDOUT** Defines a sequential data set where bind commands are written. The commands written to BINDOUT are those which need to be processed, i.e., the necessary binds.
- BINDREJ** Defines a sequential data set where bind commands are written. The commands written to BINDREJ are those which need not be processed, i.e., the unnecessary binds.
- DBRMLIB** Defines a partitioned data set containing the DBRMs for the programs to be promoted. Promote Verify reads this data set to obtain the consistency tokens that will be compared to those in the DB2 catalog.
- BINDIN** Defines a sequential data set containing the input bind control commands. These are the commands that Promote Verify will analyze and separate into necessary and unnecessary categories. The necessary commands (those which must be processed by DB2) are written to the data set identified by BINDOUT; unnecessary commands are written to BINDREJ.



Appendix B – Messages and Codes

Return Codes

All Avoid Bind messages and diagnostics are sent to SYSPRINT. Upon normal completion, you'll see one of the codes shown in Table 1. The codes are the same whether you are using traditional Avoid Bind (program \$AVB) or Coprocessor Avoid Bind (program AVBI).

Return Code	Meaning
0	Avoid Bind has completed successfully. The new source has passed SQL validation and a DB2 bind is <i>not</i> required.
4	Avoid Bind has determined that either SQL validation has failed or an error has occurred. Diagnostic messages have been issued. A fresh bind <i>is</i> required.

Table 1 — AVB Return Codes

The following table describes the return codes for Promote Verify.

Return Code	Meaning
0	Timestamps match; No bind required.
4	Timestamps disagree, a Bind is required.
8	User error, e.g., an invalid command was entered
12	Severe error, e.g., unable to connect to the DB2 subsystem

Table 2 — Promote Verify Return Codes

Your change control procedure should test the return code from Promote Verify. If the return code is zero, the application may be promoted without a bind. If the return code is anything other than zero, a bind should be performed.



Messages

Avoid Bind messages are in the form AVB0nnt; where *Onn* is the message number and *t* is a one-character suffix with the value I, E, or S. The suffix values and their meaning are as follows:

Suffix	Class	Meaning
I	Informational	Routine notification — no action required.
E	Error	Processing unable to continue for the reason given in the message text. Probable user error.
S	Severe	Environmental problem or program logic error. Continued execution is impossible.

Table 3 — AVB Message Severities

Avoid Bind will make every attempt to recover from most error situations. Under extreme conditions, such as hardware and detected logic errors, the program will abend. In these situations you should validate the old DBRM library before restarting the job.

Error Messages and Explanations

AVB0011 **Avoid Bind started for DBRM=xxxxxxx VvRr/ssssss/yy.ddd**

Explanation: Issued during Avoid Bind initialization. *xxxxxxx* is the DBRM name, *sssss* is the serial number of the CPU on which the product is executing, *yy.ddd* is the expiration date of this copy of Avoid Bind, and *v* and *r* are the version and release, respectively.

AVB0021 **Old DBRM #: TIMESTAMP: CHAR=yyyy.mm.dd.hh.mm.ss.xxxx
HEX=hhhhhhh h hhhhhhhh, DECIMAL=dddddddd ddddddd
USERID=uuuuuuuu VERSION=tttttt**

Explanation: Issued after the old (preexisting) DBRM has been located and successfully processed. The timestamp information is displayed in character format, hex(h) and decimal(d) along with the ID of the last user to precompile. The VERSION information is included in the message only if the DBRM was created with the VERSION parm. The version token *tttttt* can be up to 64 characters in length.



AVB003I **New DBRM information: TIMESTAMP: CHAR=yyyy.mm.dd.hh.mm.ss.xxxx
HEX=hhhhhhh h hhhhhhhh, DECIMAL=dddddddd dddddddd USERID=uuuuuuuu
VERSION=ttttttt**

Explanation: Issued after the new DBRM is successfully created by the DB2 precompiler, this message is similar to AVB002I, but shows the newly created timestamp and the ID of the current user, *i.e.*, the user running the job. The VERSION information is included in the message only if the DBRM was created with the VERSION parm. The version token *ttttttt* can be up to 64 characters in length.

AVB004I **Processing completed successfully, bind not required**

Explanation: Issued at end-of-job when Avoid Bind determines that a bind is not required. The old DBRM and subsequent source updates are compatible. This message is accompanied by condition code 0. The load module may be executed without a DB2 bind.

AVB005E **DBRM compare failed, source modifications have changed the content of the DBRM**

Explanation: Updates to the source code have resulted in a change to the SQL structure. A DB2 bind will be required. This message is followed by a dump of both the old and new DBRM records in which the mismatch occurred.

AVB006E **xxxxxxx DCB failed to open**

Explanation: Avoid Bind was unable to open the DD name *xxxxxxx*. Correct the DD specification and rerun.

AVB007E **sss Time stamp has invalid format, "LEVEL" precompiler option not supported**

Explanation: Either the old or new (*sss*) DBRM was generated using the DB2 precompiler LEVEL option. Avoid Bind does not support this option. If *sss* is "old" then delete the old DBRM from library pointed to by the DBRMLIB DD. If *sss* is "new" then remove the LEVEL option from the PARM= string on the EXEC statement.

AVB008E **Language type "ttttttt" is not supported**

Explanation: The HOST precompiler parameter specified a language, *ttttttt*, that is not supported by Avoid Bind. Supported language types are COB2, COBOL, PLI and ASM.

AVB009E **Avoid Bind not authorized or has expired on this CPU, serial#=nnnnnn**

Explanation: Avoid Bind was unable to locate a valid password for this CPU. *nnnnnn* is the CPU serial number for which authorization failed. Contact HLS to obtain a valid password. If you believe the password you have is valid, insure that there is not another copy of the \$AVB module located earlier in the STEPLIB or LINKLST concatenation.

AVB010S **internal error during decode or update of the new DBRM**

Explanation: Avoid Bind has encountered an internal processing error during DBRM update. The contents of the new DBRM are unpredictable. Retain the SYSUDUMP and contact HLS.



AVB011E DBRM exceeds the 3,000 SQL call restriction

Explanation: Avoid Bind has detected a DBRM containing more than 3,000 SQL calls. This exceeds the maximum supported program size.

AVB012E Syntax error or unrecognizable operand in the precompiler parameter

Explanation: Avoid Bind has detected a syntax error in the DB2 precompiler PARM=string.

AVB013E Unable to allocate the temporary DBRM save dataset

Explanation: Avoid Bind was unable to allocate a temporary DASD data set for its work file. Determine if sufficient space is available in the temporary storage pool (usually PUBLIC). If the error is due to a security violation, contact HLS for a unit name override.

AVB014E Return code from the SQL precompiler > 4

Explanation: The DB2 Precompiler ended with a return code greater than 4. Correct the coding error(s) and rerun.

AVB015E DBRMLIB dataset has an LRECL other than 80

Explanation: Avoid Bind (and the DB2 precompiler) require that DBRM libraries have a record length of 80 bytes (LRECL=80). Ensure that the library referenced by the DBRMLIB DD statement conforms to this requirement, and is the library containing the old DBRM.

AVB016E Member pointed to by the DBRMLIB DD is not a valid DBRM

Explanation: Avoid Bind has detected an invalid DBRM. This message usually indicates that the DBRMLIB DD statement is not pointing to a valid DBRM library. Check this library.

AVB017E Unable to find old DBRM member in DBRMLIB, xxxxxxxx is assumed to be new

Explanation: Avoid Bind could not locate the old DBRM member in the library pointed to by the DBRMLIB DD statement; xxxxxxxx is the DBRM name. Avoid Bind processing is bypassed for this DBRM.

AVB018S Avoid Bind has detected an Sxxx ABEND, attempting recovery

Explanation: An abend has occurred during processing. Avoid Bind is attempting to recover and continue; xxx is the system ABEND code.

AVB019E Deletion of source statements has invalidated the DBRM -- add comment lines to correct

Explanation: This condition occurs when statements are deleted from a very large source module. Due to an architectural limitation of the DB2 precompiler, the internal format of



the DBRM for a COBOL program changes when the growth of a source program forces at least one SQL statement to have a relative line number greater than 9999.

This does not pose a problem at the time it occurs, but it does cause a downward compatibility problem later on if the lines are subsequently deleted, resulting in all SQL statements once again having line numbers less than 10,000. This scenario creates a condition Avoid Bind cannot handle, so the default is to require a bind. However, if the SQL has not been modified, there is a way to circumvent this situation:

- (1) Restore the “old” DBRM (the version that was saved in the job step preceding the precompiler step).
- (2) Rather than deleting source statements, comment them out instead. This will leave the relative line numbers unchanged.
- (3) Rerun the compile job.

AVB020E DB2 Version/Release mismatch between old/new DBRMS

Explanation: This message is issued if the old and new DBRMs were created under different releases of DB2. When this occurs, an initial bind must be done under the new release; from then on Avoid Bind will function normally with the new DBRM format. To allow for possible modifications to the optimizer, it is always best to bind existing applications when the DB2 version or release level changes.

AVB021E DB2 releases greater than V8 are supported only by AVB V7R1

Explanation: This version of Avoid Bind does not support the release of DB2 that you are trying to use it with. Contact HLS to obtain a later version of Avoid Bind.

AVB022E DBRM compare failed – non matching character sets (CCSIDs)

Explanation: A mismatch was detected between the old and new DBRMs due to the fact that they were created with different character sets (CCSIDs). A bind is required.

AVB023S Avoid Bind processing terminated due to unrecoverable error

Explanation: Avoid Bind has encountered a severe error and cannot continue. This message is normally preceded by another message stating the specific error or cause of failure.

AVB024S * ERROR ** SSID DD statement is present but no subsystem ID was specified

Explanation: DBRM Generation was requested by including an SSID DD statement in the JCL, but no subsystem name was specified; Avoid Bind thus cannot determine which DB2 system catalog to generate the DBRM from. The correct syntax is:

```
//SSIDxyz DD DUMMY
```

... where xyz is the one to four character subsystem ID. In the following example, Avoid Bind is requested to use subsystem DSN1 to generate the DBRM:

```
//SSIDDSN1 DD DUMMY
```



AVB025I **Unable to extract DBRM from catalog, symptoms= xxxxxxxx /xxxxxxx**

Explanation: DBRM Generation was requested but Avoid Bind was unable to extract the relevant information from the catalog. *xxxxxxx* represents diagnostic information in hexadecimal.

AVB026I **The "old" DBRM was found in plan nnnnnnnn in the DB2 catalog**

Explanation: DBRM Generation was performed. The DBRM was reconstructed based on the DBRM in plan nnnnnnnn.

AVB027I **The "old" DBRM was found in version nnnnnnnn in the DB2 catalog**

Explanation: DBRM Generation was performed. The DBRM was reconstructed based on the package in version nnnnnnnn.

AVB098S **Avoid Bind processing terminated, mmmmmmm!!**

Explanation: this message is issued during termination when it has been determined that a DB2 bind is required. This message is normally accompanied by return code 4. *mmmmmmmm* in the message text is replaced by the text "BIND REQUIRED" unless the message is preceded by AVB014E, in which case it is replaced by "CORRECT PRECOMPILER ERRORS".

AVB099S **Avoid Bind processing failed during DBRM update, delete the DBRM and rerun**

Explanation: Avoid Bind has failed during the DBRM update. This message should be accompanied by a U999 ABEND and SYSUDUMP. The DBRM may have been corrupted. Bypass Avoid Bind processing for this particular program until a resolution to this problem can be developed. Contact HLS as soon as possible.

AVB2401S **Module mmmmmmmm could not be loaded, reason=nnn**

Explanation: Promote Verify was unable to access program *mmmmmmmm*. *nnn* is a diagnostic code describing the failure.

AVB2402E **No subsystem name has been specified**

Explanation: A valid SSID parameter was not specified. Promote Verify does not know which DB2 subsystem to query.

AVB2403S **Unable to connect to subsystem ssss, plan Promote Verify**

AVB2403S **RC=xxxxxxx, reason=yyyyyyy**

Explanation: Promote Verify attempted to connect to subsystem *ssss* but was unable to do so. *xxxxxxx* and *yyyyyyy* are diagnostic codes describing the type of failure.



AVB2404S Invalid data in SYSIBM.SYSCOLUMNS, symptom=*nnn*

Explanation: Data retrieved from the DB2 catalog is not valid. *nnn* is a diagnostic code.

AVB2405I Connected to subsystem *ssss*

Explanation: Informational message. Promote Verify has established a connection to the DB2 subsystem named *ssss*.

AVB2406S Unable to disconnect from subsystem *ssss*

AVB2406S RC=*xxxxxxxx*, reason=*yyyyyyyy*

Explanation: An error occurred when Promote Verify attempted to disconnect from subsystem *ssss*. *xxxxxxxx* and *yyyyyyyy* are diagnostic codes describing the type of failure.

AVB2407I Disconnected from subsystem *ssss*

Explanation: Informational message. Promote Verify successfully disconnected from the named subsystem.

AVB2408E *pppp* may only be specified once

Explanation: An otherwise valid Promote Verify parameter was specified multiple times. *pppp* is the name of the redundant parameter (the parameter that was specified more than once).

AVB2410E *pppp* parameter exceeds permitted length

Explanation: The value supplied for parameter *pppp* is longer than the maximum allowed.

AVB2411E *pppp* parameter missing or invalid

Explanation: Parameter *pppp* is not present or not coded correctly.

AVB2412I DB2 Version 7 or earlier

AVB2412I DB2 Version 8 or later

Explanation: Informational. One of these two messages is displayed after successfully connecting to DB2.

AVB2413E Invalid combination of parameters



Explanation: The combination of parameters specified is not valid. The only valid combinations are:

SSID, PLAN, and DBRM
or
SSID, COLLID, and PACKAGE

Any other combination is invalid.

AVB2414E Too few parameters specified

Explanation: An insufficient number of parameters was specified. Promote Verify does not have enough information to determine which catalog entry to check.

AVB2415S Unexpected EOD -- DBRM header

Explanation: The data pointed to by DD name DBRMIN is not a valid DBRM.

AVB2416S Unexpected EOD -- DBRM header extension

Explanation: The data pointed to by DD name DBRMIN is not a valid DBRM.

AVB2417S Unable to open cursor for SYSIBM.tablename, sqlcode=nnn

Explanation: An SQL error has occurred when attempting to access a table. *tablename* is the table that could not be accessed and *nnn* is a diagnostic code.

AVB2418S FETCH failed for SYSIBM.tablename, sqlcode=nnn

Explanation: An SQL error has occurred while attempting to access a table. *tablename* is the table being accessed and *nnn* is a diagnostic code.

AVB2419S Bind for Promote Verify is not current; rebind plan

Explanation: The Promote Verify bind is out of date. Rebind the program.

AVB2420I SQL diagnostics triggered at tracepoint *n*

Explanation: This message is issued following message AVB2417S, AVB2418S, or AVB2431S. It provides additional diagnostic information to help HLS Technical Support isolate and identify the problem.

AVB2421I Consistency tokens match, no bind required

Explanation: Informational message. No bind is required. This message is accompanied by return code 0.

AVB2422S DSNTIAR failure, RC=nnn

Explanation: An error occurred in DSNTIAR. *nnn* is a diagnostic code.



AVB2423I ***** A BIND IS REQUIRED *****

Explanation: Informational message. A bind is required. This message is accompanied by return code 4.

AVB2424W **AVB2424W Catalog entry not found for nnnnnnnn, no match attempted**

Explanation: Self-explanatory. The DBRM or package named in the Promote Verify control parameters was not found in the DB2 catalog. *nnnnnnnn* is either “plan DBRM” or “package” depending on the control parameters that were specified.

AVB2425I **Timestamp from DBRM: nnn (xxxxxxxx xxxxxxxx hex)**

Explanation: Informational message. *nnn* is the translated consistency token (timestamp) in the DBRM pointed to by DD name DBRMIN. *xxxxxxxx xxxxxxxx* is the “raw” value as actually stored.

AVB2426I **Timestamp from catalog: nnn (xxxxxxxx xxxxxxxx hex)**

Explanation: Informational message. *nnn* is the translated consistency token (timestamp) in the DB2 catalog. *xxxxxxxx xxxxxxxx* is the “raw” value as actually stored.

AVB2427I **Begin execution**

Explanation: Informational message. This message is issued when Promote Verify begins processing.

AVB2428I **End execution**

Explanation: Informational message. This message is issued when Promote Verify terminates.

AVB2429I *text*

Explanation: Information message. *text* is an image of one line of user input (either a control parameter or a comment).

AVB2430I **Version (from DBRM): vvvv**

Explanation: Informational message. *vvvv* is the package version in the DBRM pointed to by DD name DBRMIN. This message is issued only when a nonblank version is actually present.

AVB2431S **FETCH failed for SYSIBM.SYSCOLUMNS , sqlcode=nnn**



Explanation: An SQL error has occurred while attempting to access the specified catalog table. *nnn* is a diagnostic code.

AVB2432I ***** Avoid Bind trial has expired *****

Explanation: Self-explanatory. The trial period has elapsed.

AVB2433S **Installation or configuration error – call Technical Support**

Explanation: The copy of Promote Verify being executed is corrupted or was not installed properly. Call Technical Support.

AVB2499I **Error in Promote Verify - symptom=*nnn***

Explanation: An internal logic error has occurred in Promote Verify. *nnn* is a diagnostic code.

AVB250I **BIND CONTROL STATEMENTS INPUT *nnn***
AVB250I **BIND CONTROL STATEMENTS PROCESSED *nnn***
AVB250I **BIND CONTROL STATEMENTS BYPASSED *nnn***

Explanation: These informational messages display the number of bind commands that were input to AVB3 and how many were assigned to each category. In each line of the message, *nnn* is the count (number of commands). The “processed” statements are written to BNDOUT. The “bypassed” statements are written to BNDREJ.

AVB251I **MEMBER *memname* NOT FOUND IN DDNAME DBRMLIB**

Explanation: This message indicates that a DBRM required by one of the input bind commands was not found in DBRMLIB. In the message text, *memname* is the name of the missing DBRM.

AVB252I **BINDIN CONTROL CARDS DO NOT SPECIFY EXPLAIN(YES)**

Explanation: This warning message indicates that one or more of the input bind commands lack the EXPLAIN(YES) parameter. This condition does not affect AVB3 processing, but can make it difficult or impossible to use other tools to analyze and warn of potential access path changes caused by binding.



Notices

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

HLS TECHNOLOGIES, INC. 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. HLS may make improvements and/or changes in the product(s) and or the program(s) described in this publication at any time without notice.

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

Trademarks

IBM, DB2, MVS, ISPF, OS/390 and z/OS are trademarks of IBM Corporation.