PeopleSoft

EnterpriseOne JDE5 Development Standards Business Function Programming PeopleBook

May 2002

EnterpriseOne JDE5
Development Standards Business Function Programming PeopleBook
SKU JDE5EBS0502

Open Source Disclosure

Table of Contents

Business Function Programming Standards

A business function is an integral part of the J.D. Edwards software tool set. A business function allows application developers to attach custom functionality to application and batch processing events.

Why Have Standards for Programming Business Functions?

Since a business function is the only code not generated by J.D. Edwards software, it is important to establish firm rules regarding their makeup and usage.

In general, a standard method for creating C code that performs a specific action does not exist. C code can be as unique as the individual who programmed and compiled the code. As long as the compiled function runs, the programmer is satisfied. However, the approach a programmer uses is not always the most efficient or cleanest set of code.

This guide provides standards for coding business functions that are efficient and easy to maintain.

Technical Specifications

Program Flow

The program flow of a C function should be from the top down, modularizing the code segments. For readability and maintenance purposes, divide code into logical chunks as much as possible. In that respect, think of designing an RPG program, wherein each subroutine performs a discrete task.

Function Types

You should be familiar with two types of functions:

- Business function or J.D. Edwards business function
- Function or internal function

Business Function

A business function is also referred to as a J.D. Edwards business function because it is partially created using the J.D. Edwards software tools.

A business function is checked in and checked out so that it is available to other applications and business functions.

Application programmers spend most of their time writing J.D. Edwards business functions rather than internal functions.

Function

A function is also referred to as in internal function because it can only be used within the source file. No other source file should access an internal function.

Functions are created for modularity of a common routine that is called by the business function.

Design Standards

Business Function Design Standards are J.D. Edwards specifications for coding business functions using C programming language.

Naming Conventions

Standardized naming conventions ensure a consistent approach to identifying function objects and function sections.

Source and Header File Names

Source and header file names can be a maximum of 8 characters and should be formatted as follows:

Bxxyyyyy

Where:

B = Source or header file

xx (second two digits) = the system code, such as

01 - Address Book

04 - Accounts Payable

yyyyy (the last five digits) = a sequential number for that system code, such as

00001 - the first source or header file for the system code

00002 - the second source or header file for the system code

Both the C source and the accompanying header file should have the same name.

The following table shows examples of this naming convention.

System	System Code	Source Number	.C Source Name	Header File
Address Book	01	10	B0100010.C	B0100010.H
Accounts Receivable	04	58	B0400058.C	B0400058.H
General Ledger	09	2457	B0902457.C	B0902457.H

Function Names

An internal function can be a maximum of 42 characters and should be formatted as follows:

IBxxxxxxx_a

Where:

I = internal function

Bxxxxxxx = the source file name

a = the function description. Function descriptions can be up to 32 characters in length. Be as descriptive as possible and capitalize the first letter of each word, such as ValidateTransactionCurrencyCode. When possible, use the major table name or purpose of the function.

Example: IB4100040_CompareDate

Variable Names

Variables are storage places in a program and can contain numbers and strings. Variables are stored in the computer's memory. Variables are used with keywords and functions, such as char and MATH_NUMERIC, and must be declared at the beginning of the program.

A variable name can be up to 32 characters long. Be as descriptive as possible and capitalize the first letter of each word.

You must use Hungarian prefix notation for all variable names, as shown in the following table:

Prefix	Description
c	char
sz	NULL-terminated string
n	short
l	long
b	Boolean
mn	MATHNUMERIC
jd	JDEDATE
lp	long pointer
i	integer
by	byte
w	unsigned WORD

ul	unsigned long (identifier)
us	unsigned Short
dsxxxxxxxx	Non-J.D. Edwards tool data structures (where xxxxxxxx is source file name)
x,y	short coordinates
cx,cy	short distances
e	JDEDB_RESULT
h	handle
e	enumerated types

Example: Hungarian Notation for Variable Names

The following variable names use Hungarian notation:

char	cPaymentRecieved;
char	szCompanyNumber = "00000";
short	nLoopCounter;
long int	lTaxConstant;
BOOL	bIsDateValid;
MATH_NUMERIC	mnAddressNumber;
JDEDATE	jdGLDate;
LPMATH_NUMERIC	lpAddressNumber;
int	iCounter;
char	byOffsetValue;
unsigned word	usCalculationNumber;
unsigned long	ulFunctionStatus;
DS7037	dsInputParameters;
JDEDB_RESULT	eJDEDBResult;

Business Function Data Structure Names

The data structure for business function event rules and business functions should be formatted as follows:

DxxxxyyyyA

Where:

D = data structure

xxxx = the system code

yyyy = a next number (the numbering assignments follow current procedures in the respective application groups)

A = an alphabetical character (such as A, B, C, and so on) placed at the end of the data structure name to indicate that a function has multiple data structures

See Also

- *Creating a Business Function Data Structure* in the *Development Tools Guide*

Named Event Rule Variable Names

Event rule (ER) variables are named similar to C variables and should be formatted as follows:

xxx_yyzzzzzz_AAAA

Where:

xxx = J.D. Edwards software automatically assigns the prefix for scope, such as:

evt_

y = Hungarian Notation for C variables:

c - Character

h - handle request

mn - Math Numeric

sz - String

jd - Julian Date

id - Pointer

zzzzzz = programmer-supplied variable name; capitalize each word

AAAA = Data Dictionary alias (all upper case)

For example, a Branch/Plant event rule variable would be evt_szBranchPlant_MCU. Do not include any spaces.

Code Appearance Standards

Standards for the appearance of code ensure a consistent approach to writing code. This results in code that is readable and easily maintained.

Declaring Variables

Variables store information in memory that is used by the program. Variables can store strings of text and numbers.

The following information covers standards for declaring variables in business functions and includes examples of both standard and nonstandard formats.

Use the following checklist when declaring variables:

- Declare variables using the following format:

```
datatype       variable name = initial value;     /* descriptive comment*/
```

- Declare all variables used within business functions and internal functions at the beginning of the function. Although C allows you to declare variables within compound statement blocks, this standard requires all variables used within a function to be declared at the beginning of the function block.
- Declare only one variable per line, even if multiple variables of the same type exist. Indent each line three spaces and left align the data type of each declaration with all other variable declarations. Align the first character of each variable name (variablename in the format example above) with variable names in all other declarations. See the following example.
- Use the naming conventions set forth in this guide. When initializing variables, the initial value is optional depending on the data type of the variable. Generally, all variables should be explicitly initialized in their declaration. For further detail on initializing variables, see *Initializing Variables Initialization* in the *Business Function General Guidelines* topic.
- The descriptive comment is optional. In most cases, variable names are descriptive enough to indicate the use of the variable. However, provide a comment if further description is appropriate or if an initial value is unusual.
- Left align all comments.
- Always initialize all pointers to NULL and include an appropriate type call at the declaration line.
- Initialize all variables, except data structures, in the declaration.
- Memset all declared data structures to NULL when initialized to remove erroneous values.
- MATH_NUMERIC variables must ParseNumberString to zero. ParseNumberString to zero clears and resets the variable, which prevents MATH_NUMERIC functions from returning a general protection fault (GPF) error that can occur when MATH_NUMERIC variables contain erroneous data.

Example: C Code that Complies with the Standard for Declaring Variables

The following example complies with the standard for declaring variables:

```
JDEBFRTN  (ID) JDEBFWINAPI F0902GLDateSensitiveRetrieval
                           (LPBHVRCOM      lpBhvrCom,
                            LPVOID         lpVoid,
                            LPDSD0051      lpDS)
{
    /************************************************************
     *   Variable declarations
     ************************************************************/
    ID              idReturn         = ER_SUCCESS;
    JDEDB_RESULT    eJDEDBResult     = JDEDB_PASSED;
    long            lDateDiff        = 0L;
    BOOL            bAddF0911Flag    = TRUE;
    MATH_NUMERIC    mnPeriod;

    /************************************************************
     *  Declare structures
     ************************************************************/
    HUSER           hUser            = (HUSER) 0L;
    DSD5100016      dsDate;
    JDEDATE         jdMidDate;

    /************************************************************
     * Pointers
     ************************************************************/
    LPX0051_DSTABLES lpdsTables = (LPX0051_DSTABLES) 0L;
```

Example: C Code that Violates the Standard for Declaring Variables

The following MyBusinessFunction example does *not* comply with the standard for declaring variables:

```
/************************************************************************
*
 * NONSTANDARD

************************************************************************
/
ID  MyBusinessFunction ( LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSNNNN lpDS)
```

```
{
ID  idReturn = BHVR_SUCCESS;    /* assume success */

F9860 dsF9860;

 DS1234  ds1234;

int i,nJDEDBReturm = JDEDB_PASSED;    /* return value from JDEDB API calls
*/

   / * processing has already occurred. we're in the middle of one function
*/

   for ( i = 0; I < APREVIOUSLYDEFINEDLIMIT; i++ )

   {

         MATH_NUMERIC mnAddressNumber;

/* some more processing happens here */

}

/* rest of code follows */
```

Using Standard Variables

The requirements for standard variables are as follows:

- Any true/false flag used must be a Boolean type (BOOL).
- Name the flag variable to answer a question of TRUE or FALSE.

Flag Variables

Flag variables are listed below, with a brief description of how each is used:

bIsMemoryAllocated	Apply to memory allocation
bIsLinkListEmpty	Link List

Input Parameters

Input parameters may use the following for error messages:

cReturnPointer	When allocating memory and returning GENLNG
cCallType	Instructs when to set the error message; when a fetch fails or is successful and returns a cErrorCode
cErrorCode	Based on cCallType, cErrorCode returns a 1 when it fails or 0 when it succeeds.
cSuppressErrorMessage	If the value is 1, do not display error message using jdeErrorSet(...). If the value is 0, display the error.
szErrorMessageID	If an error occurs, return an error message ID (value); otherwise return four spaces.

Fetch Variables

Use fetch variables to retrieve and return specific information, such as a result, to define the table ID, and to specify the number of keys to use in a fetch.

eJDEDBResult	APIs or J.D. Edwards functions, such as JDEDB_RESULT
idReturnValue	Business function return value, such as ER_WARNING or ER_ERROR
idTableXXXXID	Where XXXX is the table name, such as F4101 and F41021, this is the variable used for defining the Table ID.
idIndexXXXXID	Where XXXX is the table name, such as F4101 or F41021, this variable is used for defining the Index ID of a table.
iXXXXNumColToFetch	Where XXXX is the table name, such as F4101 and F41021, this is the number of the column to fetch. *Do not* put the literal value in the APIs functions as the parameter.
iXXXXNumOfKeys	Where XXXX is the table name, such as F4101 and F41021, this is the number of keys to use in the fetch.

Example: Using Standard Variables

The following example illustrates the use of standard variables:

```
/**************************************************************************
    *  Variable declarations

**************************************************************************/
    JDEDB_RESULT   eJDEDBResult     = JDEDB_PASSED;
    ID             idTableF0901     = ID_F0901;
```

```
    ID              idIndexF0901     = ID_F0901_ACCOUNT_ID;
    ID              idFetchCol[]     = { ID_CO, ID_AID, ID_MCU, ID_OBJ,
                                         ID_SUB, ID_LDA, ID_CCT };
    ushort          nNumColToFetch   = 7;


/************************************************************************
    *  Structure declarations
************************************************************************/
    KEY3_F0901     dsF0901Key;
    DSX51013_F0901 dsF0901;


/************************************************************************
    *  Main Processing
************************************************************************/
   /** Open the table, if it is not open **/
   if ((*lpdsInfo->lphRequestF0901) == (HREQUEST) NULL)
   {
      if ( (*lpdsInfo->lphUser) == (HUSER) 0L )
      {
         eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                                      &lpdsInfo->lphUser,
                                      (char *) NULL,
                                      JDEDB_COMMIT_AUTO);
      }
      if (eJDEDBResult == JDEDB_PASSED)
      {
         eJDEDBResult = JDB_OpenTable( (*lpdsInfo->lphUser),
                                      idTableF0901,
                                      idIndexF0901,
                                      (LPID)(idFetchCol),
                                      (ushort)(nNumColFetch),
                                      (char *) NULL,
                                      &lpdsInfo->hRequestF0901 );
```

```
      }
   }
   /** Retrieve Account Master - AID only sent **/
   if (eJDEDBResult == JDEDB_PASSED)
   {
      /** Set Key and Fetch Record **/
      memset( (void *)(&dsF0901Key), (int) '\0', sizeof(KEY3_F0901) );
      strcpy ((char *) dsF0901Key.gmaid, (const char*) lpDS->szAccountID );
      eJDEDBResult = JDB_FetchKeyed ( lpdsInfo->hRequestF0901,
                                      idIndexF0901,
                                      (void *)(&dsF0901Key),
                                      (short)(1),
                                      (void *)(&dsF0901),
                                      (int)(FALSE) );
      /** Check for F0901 Record **/
      if (eJDEDBResult == JDEDB_PASSED)
      {
         ...
      }
   }
```

Using Define Statements

A define statement is a directive that sets up constants at the beginning of the program. A define statement always begins with a pound sign (#).

Use define statements sparingly. Due to the need for define statements, a separate system include header file is created for application programmers. All required define statements are in system include header files.

If a define statement is not appropriate for inclusion in a system include header file but it is required for a specific function, then include the define statement in uppercase letters within the header file for the function. An example of this follows; it uses the StartFormDynamic instruction to call a separate form.

Example: Using StartFormDynamic to Call a Separate Form

The following example includes define statements within a business function. This example uses the StartFormDynamic function to dynamically call a separate form.

```
#define          VENDOR_INFORMATION_APPLICATION     "P0401"
#define          FORM245151ORDINAL                                  2
#define          FORM245152ORDINAL                                  3
```

Using Typedef Statements

The standard for using Typedef Statements is the same as that for Define Statements. When using typedef statements, always name the object of the typedef statement using a descriptive, uppercase format.

If you are using a typedef statement for data structures or unions, remember to include the name of the business function in the name. See the following example for using a typedef statement for a data structure.

Example: Using Typedef for a User-Defined Data Structure

The following is an example of a user-defined data structure:

```
/**************************************************************************
*
 * Structure Definitions

**************************************************************************
/


typedef struct
{
   HUSER          hUser;            /** User handle **/
   HREQUEST       hRequestF0901;    /** File Pointer to the Account Master
**/
   DSD0051        dsData;           /** X0051 - F0902 Retrieval **/
   int            iFromYear;        /** Internal Variables **/
   int            iCurrentYear;
   int            iThruYear;
   int            iThruPeriod;
   BOOL           bProcessed;
   BOOL           bRollForward;
   BOOL           bRetrieveBalance;
   MATH_NUMERIC   mnCalculatedAmount;
   MATH_NUMERIC   mnCalculatedUnits;
   MATH_NUMERIC   mnChangeAmount;
   MATH_NUMERIC   mnChangeUnits;
   char           szSummaryJob[13];
   char           cSummaryLOD;
   char           cProjectionAuditTrail;
   JDEDATE        jdStartPeriodDate;
} DSX51013_INFO, *LPDSX51013_INFO;
```

Creating Function Prototypes

Refer to the following checklist when defining function prototypes:

- Always place function prototypes in the header file of the business function in the appropriate prototype section. See *Business Function Prototypes* in the *Standard Source Header* section.
- Include function definitions in the source file of the business function, preceded by a function header. See *Standard Header*.
- Ensure that function names follow the naming convention defined in this guide.
- Ensure that variable names in the parameter list follow the naming convention defined in this guide.
- List the variable names of the parameters along with the data types in the function prototype.
- List one parameter per line so that the parameters are aligned in a single column. See the following example.
- Do not allow the parameter list to extend beyond the visible page in the function definition. If the parameter list must be broken up, the data type and variable name must stay together. Align multiple line parameter lists with the first parameter. See the following example.
- Include a return type for every function. If a function does not return a value, use the keyword "void" as the return type.
- Use the keyword “void” in place of the parameter list if nothing is passed to the function.

Example: Creating a Business Function Prototype

The following is an example of a standard business function prototype:

```
/*****************************************************************************
*
 *  Business Function:  BusinessFunctionName
 *
 *        Description:  Business Function Name
 *
 *         Parameters:
 *           LPBHVRCOM           lpBhvrCom   Business Function
Communications
 *           LPVOID              lpVoid      Void Parameter - DO NOT USE!
 *           LPDSD51013          lpDS        Parameter Data Structure
Pointer
 *

*****************************************************************************
/
```

```
JDEBFRTN (ID) JDEBFWINAPI BusinessFunctionName
                          (LPBHVRCOM    lpBhvrCom,
                           LPVOID       lpVoid,
                           LPDSXXXXXX   lpDS)
```

Example: Creating an Internal Function Prototype

The following is an example of a standard internal function prototype:

```
Type XXXXXXXX_AAAAAAAA( parameter list ... );


type     : Function return value
XXXXXXXX : Unique source file name
AAAAAAAA : Function Name
```

Example: Creating a Business Function Definition

The following is an example of a standard business function definition:

```
/*
 *   see sample source for standard business function heading
*/
ID GetAddressBookDescription( LPBHVRCOM lpBhvrCom, LPVOID lpVoid,
                              LPDSNNNNNN lpDS)
{
   ID idReturn = ER_SUCCESS;
   /*------------------------------------------------------------------------
--
    * business function code
    */
   return idReturn;
}
```

Example: Creating an Internal Function Definition

The following is an example of a standard internal function definition:

```
/*-------------------------------------------------------------------------------
----
 *   see sample source for standard function header
*/
void Ib4100040_GetSupervisorManagerDefault( LPBHVRCOM lpBhvrCom, LPSTR
lpszCostCenterIn,
```

```
                                                LPSTR lpszManagerOut, LPSTR
lpszSupervisorOut )
/*-------------------------------------------------------------------------------
----
 * Note: b4100040 is the source file name
 */
{
   /*
    * internal function code
    */
}
```

Indenting Code

Any statements executed inside a block of code should be indented within that block of code.

Standard indentation is three spaces.

Note

Set up the environment for the editor you are using to set tab stops at 3 and turn the tab character off. Then, each time you press the tab key, three spaces are inserted rather than the tab character. Turn on auto-indentation.

Example: Indenting Code

The following is the standard method to indent code:

```
function block
{
   /* -----------------------------------------------------------------------------
------
    *   Any statements belong to main function block are indented three spaces from
    *      first brace
    */
   strcpy( szString1, szString2);
   /*-------------------------------------------------------------------------------
-----
    *   NOTE: If a statement starts another block of code (i.e. selection or control
    *      statements), the beginning brace lines up with the selection or control
    *      statement.
    *   Statements within the block are indented.
```

```
    */
   if ( nJDEDBReturn == JDEDB_PASSED )
   {
      CallSomeFunction( nParameter1, szParameter2 );
      CallAnotherFunction( lSomeNumber );
      while( FunctionWithBooleanReturn() )
      {
          CallYetAnotherFunction(  cStatusCode );
      }
   }
}
```

Formatting Compound Statements

Compound statements are statements followed by one or more statements enclosed with braces. A function block is an obvious example of a compound statement. Control statements (while, for) and selection statements (if, switch) are also examples of compound statements. Control and selection statements are the main subject of this section.

Refer to the following checklist when formatting compound statements:

- Always have one statement per line within a compound statement.
- Always use braces to contain the statements that follow a control statement or selection statement.
- Braces should be aligned with the initial control or selection statement.
- Logical expressions evaluated within a control or selection statement should be broken up across multiple lines if they do not fit on one line. When breaking up multiple logical expressions, do not begin a new line with the logical operator. The logical operator must remain on the preceding line.
- When evaluating multiple logical expressions, use parentheses to explicitly indicate precedence.
- Never declare variables within a compound statement, except function blocks.
- Use braces for all compound statements.

Reasons to Use Braces for All Compound Statements

Omitting braces is a common C coding practice when only one statement follows a control or selection statement. However, you must use braces for all compound statements for the following reasons:

- The absence of braces can cause errors.
- Braces ensure that all compound statements are treated the same way.
- In the case of nested compound statements, the use of braces clarifies the statements that belong to a particular code block.
- Braces make subsequent modifications easier.

Example: Failing to Use Braces

The following example is nonstandard. It is confusing and may cause errors because braces are not used.

```
/*****************************************************************************
*
 * NONSTANDARD

*****************************************************************************
/
if ( a == 0 )
   if ( y == 0 )
error();
}
else
{
   z = a + y;
   function( &z );
}
```

Example: Using Braces to Clarify Flow

In the previous example, the originator of the code intends for there to be two main cases: a=0 and a != 0. In the first case, nothing is done unless y = 0. In the second case, z should be evaluated and a function should be called. However, the example actually performs something different. An else always associates with the closest matching brace, so the statements in the else block only execute if a = 0 and y != 0. The use of braces clarifies the flow and prevents the mistake.

```
if ( a == 0 )
{
if ( y == 0 )
   {
      error();
   }
}
else
{
   z = a + y;
   function( &z );
}
```

Example: Using Braces for Ease in Subsequent Modifications

The use of braces prevents mistakes when the code is later modified. Consider the following example. The original code contains a test to see if the number of lines is less than a predefined limit. As intended, the return value is assigned a certain value if the number of lines is greater than the maximum. Later, someone decides that an error message should be issued in addition to assigning a certain return value. The intent is for both statements to be executed only if the number of lines is greater than the maximum. Instead, idReturn will be set to ER_ERROR regardless of the value of nLines. If braces were used originally, this mistake would have been avoided.

```
ORIGINAL
if (nLines > MAX_LINES)
        idReturn = ER_ERROR;


MODIFIED
if (nLines > MAX_LINES)
        jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4353", (LPVOID) NULL):
        idReturn = ER_ERROR;


STANDARD ORIGINAL
if (nLines > MAX_LINES)
{
        idReturn = ER_ERROR;
}


STANDARD MODIFIED
if (nLines > MAX_LINES)
{
        jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
        idReturn = ER_ERROR;
}
```

Example: Using a Standard Format for Compound Statements

The following example shows the standard format for compound statements:

```
if ( logical expression )
{
   if ( logical expression )
   {
      statements;
```

```
   }
   statements;
}
else if ( logical expression )
{
   statements;
}
else
{
   statements;
}
switch ( value )
{
   case CASE1:
      statement;
      statement:
      break;
   case CASE2:
      statements;
      break;
   default:
      statements;
      break;
}
while ( logical expression )
{
   statements;
}
for ( initialize; logical expression; increment )
{
}
```

Example: Handling Multiple Logical Expressions

The following example shows how to handle multiple logical expressions:

```
while ( (lWorkArray[elWorkX] < lWorkArray[elWorkMAX]) &&
        (lWorkArray[elWorkX] < lWorkArray[elWorkCDAYS]) &&
        (idReturnCode == ER_SUCCESS))
```

```
{
   ...
}
```

Using Function Calls

A function can use another function. Reuse of existing functions through a function call prevents duplicate instructions. When creating functions, refer to the following checklist when using function calls:

- Always put a space between each parameter.
- If the function has a return value, always check the return of the function for errors or a valid value.
- Use jdeCallObject to call another business function.
- When calling functions with long parameter lists, the function call should not go off the end of the visible page. Break the parameter list into one or more lines, aligning the first parameter of proceeding lines with the first parameter in the parameter list. See the following example.
- Make sure the data types of the parameters match the function prototype. When intentionally passing variables with data types that do not match the prototype, explicitly cast the parameters to the correct data type.

Example: C Code that Complies with the Standard for Calling a Function

The following example complies with the standard for calling a function:

```
/** Retrieve Calculation Information **/
if ((dsInfo.bRetrieveBalance) && (idReturn == ER_SUCCESS))
{
   idReturn = X51013_RetrieveAccountBalances( lpBhvrCom,
                                              lpVoid,
                                              lpDS,
                                              &dsInfo );
}

/** Budget Changes for Methods A and R **/
if (idReturn == ER_SUCCESS)
{
   idReturn = X51013_CheckBudgetChanges( lpBhvrCom,
                                         lpVoid,
                                         lpDS,
                                         &dsInfo );
```

```
}
```

Example: C Code that Violates the Standard for Calling a Function

The following example **does not comply** with the standard for calling a function:

```
/**************************************************************************
*
 * NONSTANDARD

**************************************************************************
/
if ((dsInfo.bRetrieveBalance) && (idReturn == ER_SUCCESS))
 X51013_RetrieveAccountBalances( lpBhvrCom,lpVoid,lpDS,&dsInfo );


if (idReturn == ER_SUCCESS) X51013_CheckBudgetChanges( lpBhvrCom,
lpVoid,lpDS,&dsInfo );
```

Example: Using jdeCallObject

The following example uses jdeCallObject to call another business function:

```
   /*------------------------------------------------------------------------
    *
    *   Retrieve account master information
    *
    *------------------------------------------------------------------------
*/
   memset( (void *)(&dsValidateAccount), (int) '\0', sizeof( DSD0900028 ) );
   dsValidateAccount.cBasedOnFormat = '1';
   strcpy((char *)  dsValidateAccount.szAccountID,
                (const char *)(lpDS->szPA_PUAccountNumberShort) );


   idReturnCode = jdeCallObject("ValidateAccountNumber",
                                NULL,
                                lpBhvrCom,
                                lpVoid,
                                (void*) &dsValidateAccount,
                                (CALLMAP*) NULL,
                                (int) 0,
                                (char*) NULL,
```

```
                                    (char*) NULL,
                                    (int) 0 );


   if ( idReturnCode == ER_SUCCESS )
   {
      ...
   }
```

Including Comments

When coding functions, you should use comments to describe the purpose of the function and your intended approach. This action will make future maintenance and enhancement of the function easier.

Use the following checklist for including comments:

- Always use the /*comment */ style. The use of // comments is not portable.
- Keep comments simple and concise.
- Precede and align comments with the statements they describe.
- Comments should never go off the end of the visible page.

Example: Inserting Comments within the Source Code

```
   /*----------------------------------------------------------------------
    *  Comment blocks need to have separating lines between the text
    *  description. The separator can be a dash '-' or an asterisk '*'
    *----------------------------------------------------------------------*/


   if ( ... )
   {
      ...
   } /* inline comments are used to indicate meaning of one statement */


   /*----------------------------------------------------------------------
    *  Comments should be used in all segments of the source code.  The
    *  original programmer may not be the programmer maintaining the code
    *  in the future which makes this a crucial step in the development
    *  process.
    *----------------------------------------------------------------------
*/
```

```
/*************************************************************************
    *  Function Clean Up

*************************************************************************/
```

Example: Inserting Comments within the Header File

The following example shows the use of comments within the header file:

```
/************************************************
 * Structure Definitions
************************************************/
typedef struct {
            char   lflnty[3];
            char   lflnds[31];
            char   lflnd2[31];
            char   lfgli;
}DSSourceNameXXXXX;              /* Get Line Type Constants */
/*-------------------------------------------------------------------------------
------
 *   Put in comment if multiple data structures are defined to indicate the purpose
and
 *      where it is used
 */
typedef struct {
            char   dcto[3];
            char   lnty[3];
            char   trty[4];
}DSSourceNameYYYYY;             /* Verify Activity Rule Status Code */
/************************************************************************
 * DS Template Type Definitions
 ************************************************************************/
/*********************************
 * TYPEDEF for Data Structure
 *   Template Name: Build Status Code Index
 *   Template ID:   92042
 *   Generated:     Wed Feb 08 14:19:27 1995
 *
```

```
 * DO NOT EDIT THE FOLLOWING TYPEDEF
 *   To make modifications, use the Data Structure
 *      Tool to Generate a revised version, and paste from
 *      the clipboard.
 *
 ************************************/
#ifndef DATASTRUCTURE_92042
#define DATASTRUCTURE_92042
typedef struct tagDS92042
{
   char              szLineStatusCode[4];                   /* Line Status Code */
char              cDestroyIndexList;                    /* Destroy Index Link List
(Y/N) */
} DS92042, FAR *LPDS92042;
#define IDERRszLineStatusCode_1                  1L
#define IDERRcDestroyIndexList_2                 2L
#endif    /* Build Status Code Index */
/* ----------------------------------------------------------------------------
-------
 *   Put a comment on the #endif line to indicate the end DS template on one set.
 *   This will be helpful if there are multiple DS templates in the header, and the
template
 *      more than a page or screen long.
 */
```

Using the Function Clean Up Area

Use the function clean up area to release any allocated memory, including hRequest and hUser.

Example: Using Function Clean Up Area to Release Memory

The following example shows how to release memory in the function clean up area:

```
...
lpdsF4301 = (LPF4301)jdeAlloc( COMMON_POOL,sizeof(F4301),MEM_ZEROINIT ) ;
...
/************************************************************************
```

```
 * Function Clean Up Section
 ************************************************************************/

if (lpdsF4301 != (LPF4301 ) NULL)
{
   jdeFree( lpdsF4301 );
}


if (hRequestF4301 != (HREQUEST) NULL)
{
   JDB_CloseTable( hRequestF4301 );
}


JDB_FreeBhvr( hUser );


return ( idReturnValue ) ;
```

Business Function General Guidelines

This chapter provides general guidelines you should consider when programming J.D. Edwards business functions.

Initializing Variables

Two types of variable initialization exist: explicit and implicit. Variables are explicitly initialized if they are assigned a value in the declaration statement. Implicit initialization occurs when variables are assigned a value during the course of processing.

Refer to the following general guidelines to initialize variables:

- Generally, variables with obvious initial or default values should always be explicitly initialized. For example, return values for business functions should always be explicitly initialized to the default value of ER_SUCCESS.
- Data structures should be initialized to zero using memset immediately after the declaration section.
- Some APIs, such as the JDB ODBC API, provide initialization routines. In this case, the variables intended for use with the API should be initialized with the API routines.
- Any pointers defined must be explicitly initialized to NULL with a preceding appropriate typecast.

Example: Initializing Variables

The following example shows how to initialize variables:

```
/************************************************************************
```

```
    *  Variable declarations

*****************************************************************************/

   ID               idReturn       = ER_SUCCESS;
   JDEDB_RESULT     eJDEDBResult   = JDEDB_PASSED;
   long             lDateDiff      = 0L;
   BOOL             bAddF0911Flag  = TRUE;
   MATH_NUMERIC     mnPeriod;


/*****************************************************************************
    *  Declare structures

*****************************************************************************/
   HUSER            hUser          = (HUSER) 0L;
   HREQUEST         hRequestF0901  = (HREQUEST) 0L;
   DSD5100016       dsDate;
   JDEDATE          jdMidDate;


/*****************************************************************************
    *  Pointers

*****************************************************************************/
   LPX0051_DSTABLES lpdsTables = (LPX0051_DSTABLES) 0L;


/*****************************************************************************
    *  Check for NULL pointers

*****************************************************************************/
   if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
       (lpVoid    == (LPVOID)    NULL) ||
       (lpDS      == (LPDSD0051)     NULL))
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
      return ER_ERROR;
   }
```

```
/************************************************************************
    *  Main Processing

************************************************************************/
   eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                                &hUser,
                                (char *) NULL,
                                         JDEDB_COMMIT_AUTO);


   memset  ((void *)(&dsDate), (int)'\0', sizeof(DSD5100016) );
   memcopy ((void*)) &dsDate.jdPeriodEndDate,
             (const void*) &lpDS->jdGLDate, sizeof(JDEDATE));
```

Typecasting

Typecasting is also known as type conversion. Use typecast when the function requires a certain type of value.

All functions, including J.D. Edwards business functions, internal functions, and `C' functions, must have typecast parameters.

Note

This standard is for all function calls as well as function prototypes.

Any memory allocation using jdeAlloc( ) must be typecast with proper type.

Using Constants

Generally, do not use constants within a function. This rule has exceptions. It is appropriate to use a constant in a business function when the constant does not already exist in the system include header file and the constant has no purpose in the system include header file. An example of an exception is using a call to the StartFormDynamic function.

Comparison Testing

Always use explicit tests for comparisons.

The following examples show how to create ‘C’ code for comparison tests. For comparison, both an example of ‘C’ code that complies with the standard and an example that violates the standard are shown.

Example: C Code that Complies with the Standard for Comparison Tests

In the example that follows, the intention of the test is clear to anyone who reads or maintains this code. If at any point during development the JDEDB_PASSED value changes, then the effect is not dramatic in the 'C' function.

The only exception to this rule is if the function return value or variable being tested is truly Boolean, and the only possible values are true or false.

```
eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                              &hUser,
                              (char *) NULL,
                              JDEDB_COMMIT_AUTO);


/** Check for Valid hUser **/
if (eJDEDBResult == JDEDB_PASSED)
{
   ...
}
```

Example: 'C' Code that Violates the Standard for Comparison Tests

In the example that follows, assume there is only one value for JDEDB_PASSED. By explicitly testing for JDEDB_PASSED, it ensures that only one result exists. For any other value, the function fails.

```
/*****************************************************************************
*
 *  NONSTANDARD


*****************************************************************************
*/
eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                              &hUser,
                              (char *) NULL,
                              JDEDB_COMMIT_AUTO);


/** Check for Valid hUser **/
if (!eJDEDBResult)
{
   ...
}
```

Creating True/False Test Comparison that Uses Boolean Logic

Do not use embedded assignments for comparison tests. Embedded assignments are more difficult to read and harder to maintain.

Example: Creating TRUE/FALSE Test Comparison that Uses Boolean Logic

```
/* IsStringBlank has a BOOL return type. It will always return either TRUE or FALSE */
if ( IsStringBlank( szString) )
{
   statement;
   statement;
}
```

Embedding Assignments in Comparison Tests

Always test floating point variables as <= or >=, and never use == or != since some floating point numbers cannot be represented exactly.

The following examples contrast the preferred method with an alternate method for embedding assignments for comparison tests.

Example: Embedding Assignments in Comparison Tests - Preferred Method

The following example shows how to write embedding assignments so that they are readable:

```
if ( (nJDBReturn = JDBInitBhvr(...)) == JDEDB_PASSED)
{
   statement;
}
```

Example: Embedding Assignments in Comparison Tests - Alternate Method

The following example shows one method for embedding assignments in comparison test. While this method is allowed, it is not preferred:

```
nJDEDBReturn = JDBInitBhvr(...);
if ( nJDEDBReturn == JDEDB_PASSED )
{
   statement;
}
```

Accessing the Database Efficiently

Currently, J.D. Edwards open system applications use open database connectivity (ODBC) as its database driver. This interface is unimportant for the programmer. However, the ODBC interface is very important for the J.D. Edwards database APIs. Several issues must be considered whenever I/O to the J.D. Edwards database is performed.

When writing a new record to a database file, the entire structure must be declared and initialized before writing the new record. The J.D. Edwards APIs will write a new record without initializing undeclared fields. Any subsequent read from the file to retrieve records leaves a NULL value for uninitialized fields.

The J.D. Edwards database is an open database so the file that must be read from or written to may not be located on the machine in which the program is running. The time it takes to retrieve a record depends on:

- Network communication and the machine on which the file exists
- Transmission time required to return the requested data to the requesting machine

The only variables in this process that programmers can control are the number of requests and fields requested. Because of this, consider performance before implementing the J.D. Edwards database APIs. The issues to consider can be broken down into the following performance questions and answers.

Questions:

- How many fields must be requested at one time?
- How many records must be requested?
- How many different views (or logical keys) are required for any one file?

Answers:

- The initial open table (JDE_OpenTable) API reduces performance. Limit the number of data structures required to retrieve the information you need from a file.
- When the program writes a new record, the entire file structure must be initialized and used.
- If a large number of records must be read, requiring more than 20 I/Os, then use as few fields as possible.
- Avoid opening a table more than once to retrieve a record using the same function. If you want to retrieve records from a table based on a different set of indices, or logicals, use the JDB_FetchKeyed and JDB_SelectKeyed APIs. For example, suppose you want to fetch a record based on the primary key of the table and a record from the same table based on a different key, use JDB_FetchKeyed defining the appropriate parameter for each key in the syntax.

Inserting Function Exit Points

Where possible, use a single exit point (return) from the function. The code is more structured when a business function has a single exit point. The use of a single exit point also allows the programmer to perform cleanup, such as freeing memory and terminating ODBC requests, immediately before the return. In more complex functions, this may be difficult or unreasonable.

No rules exist for creating business functions with single exit points. J.D. Edward recommends that you continually use the return value of the function to control statement execution. For example, business functions can have one of two return values:

- ER_SUCCESS - ER processed successfully
- ER_ERROR - an error occurred and execution of subsequent business functions attached to the event stop

By initializing the return value for the function to ER_SUCCESS, the return value can be used to determine the processing flow.

Note

If the exit point is required in the middle of a function, then all memory allocated, huser, and hrequest must be released.

Example: Inserting an Exit Point in a Function

The following example demonstrates the use of a return value for the function to control statement execution:

```
   ID                  idReturn               = ER_SUCCESS;



/************************************************************************
    * Declare structures

************************************************************************/
   DSX51013_INFO       dsInfo;



/************************************************************************
    * Check for NULL pointers

************************************************************************/
   if ((lpBhvrCon == (LPBHVRCOM) NULL) ||
       (lpVoid    == (LPVOID)    NULL) ||
       (lpDS      == (LPDSD51013)    NULL))
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
      return ER_ERROR;
   }
```

```
/*************************************************************************
    * Main Processing

*************************************************************************/
   memset( (void *)(&dsInfo), (int) '\0', sizeof(DSX51013_INFO) );
   idReturn = X51013_VerifyAndRetrieveInformation( lpBhvrCom,
                                                   lpVoid,
                                                   lpDS,
                                                   &dsInfo );

   /** Check for Errors and Company or Job Level Projections **/
   if ( (idReturn == ER_SUCCESS) &&
        (lpDS->cJobCostProjections == 'Y') )
   {
      /** Process All Period between the From and Thru Dates **/
      while ( (!dsInfo.bProcessed) &&
              (idReturn == ER_SUCCESS) )
      {
         /** Retrieve Calculation Information **/
         if ((dsInfo.bRetrieveBalance) && (idReturn == ER_SUCCESS))
         {
            idReturn = X51013_RetrieveAccountBalances( lpBhvrCom,
                                                       lpVoid,
                                                       lpDS,
                                                       &dsInfo );
         }

         if (idReturn == ER_SUCCESS)
         {
            ...
         }
      } /* End Processing */
   }

/*************************************************************************
```

```
    * Function Clean Up

*************************************************************************/
   if ( (dseInfo.hUser) != (HUSER) 0L )
   {
      ...
   }


   return idReturn;
```

Calling an External Business Function

Use jdeCallObject to call an external business function defined in the Object Librarian.

Call an internal business function within the same source code. An external call for an internal business function causes unexpected results and is prohibited in the client/server architecture.

To prevent internal functions from being used outside the source code, include the word "static" to keep the scope of the function local and to place prototypes within the .c file.

Example: Calling an External Business Function

The following example calls an external business function:

```
   /*------------------------------------------------------------------
    *
    *  Retrieve account master information
    *
    *------------------------------------------------------------------*/
   memset( (void *)(&dsValidateAccount), (int) '\0', sizeof( DSD0900028 ) );
   dsValidateAccount.cBasedOnFormat = '1';
   strcpy((char *) dsValidateAccount.szAccountID,
                 (const char *)(lpDS->szPA_PUAccountNumberShort) );


   idReturnCode = jdeCallObject( "ValidateAccountNumber",
                                 NULL,
                                 lpBhvrCom,
                                 lpVoid,
                                 (void*) &dsValidateAccount,
                                 (CALLMAP*) NULL,
                                 (int) 0,
                                 (char*) NULL,
```

```
                                    (char*) NULL,
                                    (int) 0 );


   if ( idReturnCode == ER_SUCCESS )
   {
      ...
   }
```

Using GENLNG as an Address

GENLNG is a dictionary object with a long integer as a type or ID. Use GENLNG to attach an index number to an array that retains the address of the platform.

When the GENLNG Address Exceeds 32 Bits

The address can be longer than 32 bits. If you pass the address for a GENLNG that exceeds 32 bits across the platform to a client that supports 32 bits, the significant digit might get truncated. The missing digit will cause the system to respond with a GPF.

When a GENLNG is Assigned to an Array

The array for the address of the allocated memory is located on the server platform or the location specified in the configuration for business function processing. The array allows up to 100 memory locations to be allocated and stored. When you use a GENLNG in a business function, consider the following:

- Before an address is assigned to a GENLNG, the GENLNG must be initialized to 0 at the beginning of the program.
- An address is only stored in the array if the fetch and memory allocation are successful.

You can perform the following activities for a GENLNG that is assigned to an array:

- Storing an address in an array
- Retrieving an address from an array
- Removing an address from an array

Storing an Address in an Array

Use jdeStoreDataPtr to store an allocated memory pointer in an array for later retrieval. The J.D. Edwards software tools maintain the array. The jdeAlloc API must be used to allocate memory addresses because business functions reside on a client or server.

Example: Storing an Address in an Array

The following example demonstrates how to store an address in an array:

```
If (lpDS->cReturnF4301PtrFlag == '1')
{
```

```
  lpdsF4301 = (LPF4301)jdeAlloc(COMMON_POOL,sizeof(F4301),MEM_ZEROINIT);

  if (lpdsF4301 != (LPF4301) 0L)
  {
    memcpy ((void *)(lpdsF4301),(const void *)(&dsF4301Fetch,sizeof(F4301));

    lpDS->idF4301RowPtr = jdeStoreDataPtr(hUser,(void *)lpdsF4301);

    if (lpDS->idF4301RowPtr == (ID) 0L)
    {
      idReturnValue = ER_ERROR;
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "3143", (LPVOID) NULL);
      jdeFree((void *)lpdsF4301);
    }
  }/*if memory allocation was successful*/
  else
  {
    idReturnValue=ER_ERROR;
  }/*if memory allocation was unsuccessful */
}
```

Retrieving an Address from an Array

Use jdeRetrieveDataPtr to retrieve an address outside the current business function. When you use jdeRetrieveDataPtr, the address remains in the array.

Example: Retrieving an Address from an Array

The following example retrieves an address from an array:

```
   lpdsF43199 = (LPF43199) jdeRetrieveDataPtr (hUser, lpDS-
>idF43199Pointer);
   if ( lpdsF43199 != (LPF43199) NULL)
   {
      dsPurchaseLedger.idPointerToF43199DS = (ID) lpdsF43199;
   }
   else
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "3143", (LPVOID) NULL);
```

```
      eJDEDBResult = ER_ERROR;
   }
```

Removing an Address from an Array

Use jdeRemoveDataPtr to free the memory allocated for the address retrieved. When you use jdeRemoveDataPtr, it removes the address from the array cell and releases the array cell.

Example: Removing an Address from an Array

The following example removes an address from an array cell:

```
/**************************************************************************
 * Main Processing

**************************************************************************/


   if (lpDS->idGenericLong != (ID) 0)
   {
      lpGenericPtr = (void *)jdeRemoveDataPtr(hUser,lpDS->idGenericLong);


      if (lpGenericPtr != (void *) NULL)
      {
         jdeFree((void *)lpGenericPtr);
         lpDS->idGenericLong = (ID) 0;
      }
      else
      {
         idReturnCode = ER_ERROR;  /*  error retrieving ptr address  */
      }
   }
```

A business function receiving a GENLNG must check for NULL and the spec must identify the procedure when a NULL is encountered in the GENLNG.

The following rules apply to releasing an address from GENLNG:

- Never release the address in the GENLNG in the receiving business function.
- If a GENLNG is used for passing into another business function through a data structure from within your business function, it must be released right after calling function is completed and reset to NULL.
- If a GENLNG is passed into a business function from ER (Event Rules), GENLNG must be released at the ER level with Free Ptr to Data Structure.

The cReturnPointer (EV01) flag indicates whether the GENLNG should be passed back out. The following scenarios apply when using CReturnPointer (EV01):

- cReturnPointer! = `1'

  Do not return any GENLNG and do not allocate any memory associated with it.

- cReturnPointer = `1'

  If the fetch is successful, allocate memory, load the record into the allocated memory, and return the address to the GENLNG.

Allocating Memory

Use jdeAlloc to allocate memory for all data types if:

- The business function must pass back an address in GENLNG through lpDS
- The fetch is successful

Because jdeAlloc affects performance, avoid using jdeAlloc unless values stored in memory are used outside of the business function. Only allocate memory when you retrieve the entire record from a table that contains a significant number of columns. Since table columns must pass from the business function for use in other business functions, it is more efficient to allocate memory for just those columns needed rather than passing all columns from the business function.

Example: Allocating Memory

See the following example.

```
eJDEDBReturn = JDB_FetchKeyed( hRequestF4301, (ID) 0,
                                (void *)(&dsF4301Key1),
                                (short)(iF4301NumOfKeys),
                                (void *)(&dsF4301Fetch),
                                (int)(FALSE) ) ;
if ( eJDEDBReturn == JDEDB_PASSED )
{
   lpdsF4301 = (LPF4301) jdeAlloc(COMMON_POOL,sizeof(F4301),MEM_ZEROINIT) ;

   if ( lpdsF4301 != (LPF4301)NULL)
   {
          memcpy ((void *)(lpdsF4301), (const void *)(&dsF4301Fetch),
                  sizeof(F4301) ) ;
      lpDS->idF4301RowPtr = jdeStoreDataPtr(hUser,(void *)lpdsF4301);

      if (lpDS->idF4301RowPtr == (ID) 0L)
      {
         idReturnValue = ER_ERROR;
         jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "3143", (LPVOID) NULL);
```

```
         jdeFree((void *)lpdsF4301);
      }
   } /*if memory allocation was successful */
   else
   {
      idReturnValue=ER_ERROR;
   } /*if memory allocation was unsuccessful */
}
```

Releasing Memory

If memory is allocated in a business function and no address is passed out, the memory must be released in that business function within the clean up section of the code, and the address must be reset to NULL. Use one of the following options to release memory:

- Use jdeFree to release memory within a business function.
- Use the business function Free Ptr To Data Structure, B4000640, to release memory through event rule logic.

Example: Releasing Memory within a Business Function

The following example uses jdeFree to release memory in the function clean up section:

```
...
lpdsF4301 = (LPF4301)jdeAlloc( COMMON_POOL,sizeof(F4301),MEM_ZEROINIT ) ;
...


/************************************************************************
 * Function Clean Up Section
 ************************************************************************/
if (lpdsF4301 != (LPF4301) NULL)
{
   jdeFree( lpdsF4301 );
}


if (hRequestF4301 != (HREQUEST) NULL)
{
   JDB_CloseTable( hRequestF4301 );
}
```

```
JDB_FreeBhvr( hUser ) ;


return ( idReturnValue ) ;
```

Initializing Data Structures for Updating

When writing to the table, the table recognizes the following default values:

- Space-NULL if string is blank
- 0 value if math numeric is 0
- 0 JDEDATE if date is blank
- Space if character is blank

Always memset to NULL on the data structure that is passed to another business function to update a table or fetch a table.

Example: Using Memset to Reset the Data Structure to Null

The following example resets the data structure to NULL when initializing the data structure:

```
bOpenTable = B5100001_F5108SetUp( lpBhvrCom, lpVoid, lphUser,
&hRequestF5108);


if ( bOpenTable )
{
   memset( (void *)(&dsF5108Key), '\0', sizeof(KEY1_F5108) );
   strcpy( (char*) dsF5108Key.mdmcu, (const char*) lpDS->szBusinessUnit );
   memset( (void *)(&dsF5108), '\0', sizeof(F5108) );


   JDB_ClearColBuffer( hRequestF5108, (void *)(&dsF5108) );
   strcpy( (char*) dsF5108.mdmcu, (const char*) lpDS->szBusinessUnit );
   MathCopy(&dsF5108.mdbsct, &mnCentury);
   MathCopy(&dsF5108.mdbsfy, &mnYear);
   MathCopy(&dsF5108.mdbtct, &mnCentury);
   MathCopy(&dsF5108.mdbtfy, &mnYear);
   eJDEDBResult = JDB_InsertTable( hRequestF5108,
                                   ID_F5108,
                                   (ID)(0L),
                                   (void *) (&dsF5108) );
|
```

Using hRequest and hUser

You must include hUser and hRequest with an API call. If an API is not used in the business function, do not use JDBInitBhvr.

Initialize hUser and hRequest to NULL in the variable declaration line. Additionally, all hRequest and hUser declarations should have JDB_CloseTable( ) and JDB_FreeBhvr( ) in the function clean up section of the code.

Inserting Required Parameters in lpDS for Error Messages

You must insert the parameters cSuppressErrorMessage and szErrorMessageID in lpDS for an error message processing. The functionality for each is described below:

- cSuppressErrorMessage (SUPPS)

  Valid data are either a 1 or 0. This parameter is required if jdeErrorSet(...) is used in the business function. When cSuppressErrorMessage is set to 1, do not set an error. This is because jdeErrorSet will automatically display an error message.

- szErrorMessageID (DTAI)

  This string contains the error message ID value that is passed back by the business function. If an error occurs in the business function, szErrorMessageID contains that error number ID.

Example: Required Parameters in lpDS for an Error Message

The following example includes the required lpDS parameters, cSuppressErrorMessage, and szErrorMessageID:

```
   if ((!IsStringBlank(lpDS->szErrorMessage)) &&
       (lpDS->cSuppressErrorMessage != '1'))
   {
      strcpy ((char*) (lpDS->szErrorMessage), (const char*) ("0653"));
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) IDERRcMethodofComputation_1,
                   lpDS->szErrorMessage, (LPVOID) NULL);
      idReturnValue = ER_ERROR;
   }


/*************************************************************************
    * Function Clean Up

*************************************************************************/

   return idReturnValue;
```

Note

szErrorMessageID must be initialized to 4 spaces.

cCallType (EV02)

Depending on whether a fetch passes or fails, use the cCallType (EV02) flag to set the cErrorCode or szErrorMessageID. The following scenarios apply for cCallType (EV02):

- cCallType = `1'

  If the fetch fails, set cErrorCode equal to `1'; otherwise cErrorCode is equal to `0'.

- cCallType = `2'

  If the fetch is successful, set cErrorCode equal to `1'; otherwise cErrorCode is equal to `0'.

- cErrorCode (ERR)

  This flag is set to `0' or `1' based on the cCallType condition.

Portability

Portability is the ability to run a program on more than one computer without modifying it. J.D. Edwards is creating a portable environment. This chapter presents considerations and guidelines for porting objects between systems.

Business functions must be ANSI-compatible for portability. Since different computer platforms may present limitations, there are exceptions to this rule. However, do not use another non-ANSI exception without approval by the Business Function Standards committee.

Standards that impact the development of relational database systems are determined by the:

- ANSI (American National Standards Institute) standard
- X/OPEN (European body) standard
- ISO (International Standards Institute) SQL standard

Ideally, industry standards should allow users to work identically with different relational database systems. The issue is that each major vendor supports industry standards but also offers extensions to enhance the functionality of the SQL language. Vendors are also constantly releasing upgrades and new versions of their products.

These extensions and upgrades affect portability. Due to the industry impact of software development, applications need a standard interface to databases without being impacted by differences between database vendors. When vendors provide a new release, the impact on existing applications needs to be minimal. To solve many of these portability issues, many organizations are moving to standard database interfaces called open database connectivity (ODBC).

Refer to the following checklist for developing business functions that comply with portability standards:

- ❑ Do not create a program that depends on data alignment because it is not portable. This is because each system aligns data differently by allocating bytes or words. Suppose there is a one-character field that is one byte. Some systems allocate only one byte for that field, while other systems allocate the entire word for the field.

- Keep in mind that vendor libraries and function calls are system-dependent and exclusive to that vendor. This means if the program is compiled using a different compiler, that particular function will fail.
- Avoid using a pointer arithmetic because it is system-dependent and is based on the data alignment.
- Do not assume that all systems will initialize the variable the same way. Always explicitly initialize the variables.
- Avoid using the offset to retrieve the object within the data structure. This also relates to data alignment.
- Always typecast if your parameter is not matching the function parameter.

  Note: char szArray[13] is not the same as (char *) in the function declaration. Therefore, typecast of (char *) is required for szArray when used in that particular function.

- Never typecast on the left-hand side of the assignment statement. This might cause the loss of data, such as (short) nInter=(long Inter)
- *Do not use C++ comments*. (C++ comments begin with two forward slashes.) See the examples that follow.

Example: Creating a C++ Comment that Complies with the ANSI Standard

Use the following C standard comment block:

```
/* This is an example of how comments should be used */
```

Example: Creating a C++ Comment that Violates the ANSI Standard

The following method is nonstandard. Do *not* use it to create a C++ comment.

```
/*************************************************************************
 * NONSTANDARD

*************************************************************************/
//
This is a C++ comment line and an example of how not to do comments
```

J.D. Edwards Defined Structures

J.D. Edwards software provides two data types that you should be concerned with when you create business functions: MATH_NUMERIC and JDEDATE. It is always possible that these data types may change. For that reason, it is critical that the Common Library APIs provided by J.D. Edwards software are used to manipulate variables of these data types.

MATH_NUMERIC Data Type

The MATH_NUMERIC data type is used exclusively to represent all numeric values in J.D. Edwards software. The values of all numeric fields on a form or batch process are communicated to business functions in the form of pointers to MATH_NUMERIC data structures. MATH_NUMERIC is used as a Data Dictionary data type.

The data type is defined as follows:

```
struct tagMATH_NUMERIC


{
   char  String [MAXLEN_MATH_NUMERIC + 1];
   char  Sign;
   char  EditCode;
   short nDecimalPosition;
   short nLength;
   WORD  wFlags;
   char  szCurrency [4];
   short nCurrencyDecimals;
   short nPrecision;
};


typedef struct tagMATH_NUMERIC MATH_NUMERIC, FAR *LPMATH_NUMERIC;
```

MATH_NUMERIC Element	Description
String	The digits without separators
Sign	A minus sign indicates the number is negative. Otherwise, the value is 0x00.
EditCode	The Data Dictionary edit code used to format the number for display.
nDecimalPosition	The number of digits from the right to place the decimal.
nLength	The number of digits in the String.
wFlags	Processing flags.
szCurrency	The currency code.
nCurrencyDecimals	The number of currency decimals.
nPrecision	The data dictionary size.

JDEDATE Data Type

The JDEDATE data type is used exclusively to represent all dates in J.D. Edwards software. The values of all date fields on a form or batch process are communicated to business functions in the form of pointers to JDEDATE data structures. Never access fields directly for year 2000 compliance. JDEDATE is used as a Data Dictionary data type.

The data type is defined as follows:

```
struct tagJDEDATE
{
   short nYear;;
   short nMonth;;
   short nDay;
};

typedef struct tagJDEDATE JDEDATE, FAR *LPJDEDATE;
```

JDEDATE Element	Description
nYear	The year.
nMonth	The month.
nDay	The day.

Standard Header and Source Files

Model source code files exist for both the .C and .H modules of your business function. Business Function Design generates the .c and .h templates, including #include, for tables and other business functions.

Standard Header

Header files are necessary to help the compiler properly create a business function. The C language contains 33 keywords. Everything else, such as printf and getchar, is a function. Functions are defined in various header files you include at the beginning of a business function. Without header files, the compiler does not recognize the functions and may return error messages.

Standard Source Header

This chapter provides an example of standard header for a business function source file. The following is an example of a standard source header for a business function.

```
/**************************************************************************
*
 *    Header File:  BXXXXXXX.h
 *    Description:  Generic Business Function Header File
```

```
 *        History:
 *          Date        Programmer  SAR# - Description
 *          ----------  ----------  ---------------------------------------
-
 *   Author 06/06/1997                     - Created
 *
 * Copyright (c) J.D. Edwards World Source Company, 1996
 *
 * This unpublished material is proprietary to J.D. Edwards World Source
 * Company.  All rights reserved.  The methods and techniques described
 * herein are considered trade secrets and/or confidential.  Reproduction
 * or distribution, in whole or in part, is forbidden except by express
 * written permission of J.D. Edwards World Source Company.

****************************************************************************
/


#ifndef __BXXXXXXX_H
#define __BXXXXXXX_H
/****************************************************************************
*
 * Table Header Inclusions

****************************************************************************
/


/****************************************************************************
*
 * External Business Function Header Inclusions

****************************************************************************
/


/****************************************************************************
*
 * Global Definitions

****************************************************************************
/
```

```
/******************************************************************************
*
 * Structure Definitions

******************************************************************************
/


/******************************************************************************
*
 * DS Template Type Definitions

******************************************************************************
/


/******************************************************************************
*
 * Source Preprocessor Definitions

******************************************************************************
/
#if defined (JDEBFRTN)
       #undef JDEBFRTN
#endif


#if defined (WIN32)
       #if defined (WIN32)
              #define JDEBFRTN(r) __declspec(dllexport) r
       #else
              #define JDEBFRTN(r) __declspec(dllimport) r
       #endif
#else
       #define JDEBFRTN(r) r
#endif
/******************************************************************************
*
 * Business Function Prototypes

******************************************************************************
/
JDEBFRTN (ID) JDEBFWINAPI GenericBusinessFunction
```

```
                 (LPBHVRCOM      lpBhvrCom,
                  LPVOID         lpVoid,
                  LPDSDXXXXXXXX  lpDS);


/*****************************************************************************
*
 * Internal Function Prototypes

*****************************************************************************
/
#endif    /* ___BXXXXXXX_H */
```

Header Sections

The standard source header is broken up into various sections.

Business Function Name and Description

Use the #include section to define the name and a brief description of the business function. Also use this section to maintain the modification log.

- For Header File: change example.c to your source member name.
- For Description: include a brief but clear explanation of what your business function accomplishes.
- Include a line with the date you first created the business function or checked out an existing business function and your initial and last name. Also include on the same line either the notation "Created" for a new business function or the SAR number for an enhancement or bug fix.

Example: Defining the Name and Description of a Business Function

The following example uses #include for defining the name and description of a business function:

```
#include
/*****************************************************************************
 * Header File:  example.h
 *
 * Description:  Example of ".h" member for coding standards.
 *         See bfstdhdr.c on the server in \b7\ins for template.
 *
 *       History:                  Programmer               SAR# -
Description
 *         Date
 *        --------------------   ----------------------  -------------------
--------
```

```
 * Author  12/01/1994                  MM247887             Unknown -
Created
*****************************************************************************/
```

Copyright Notice

The Copyright section contains the J.D. Edwards & Company copyright notice and must be included in each source file. Do not make any changes to this section.

Example: Copyright Section

The following is an example of the standard copyright section:

```
/*****************************************************************************
**
* Copyright (c) 1994
* J.D. Edwards & Company
*
* This unpublished material is proprietary to  J.D. Edwards & Company.
* All rights reserved.  The methods and techniques described herein are
* considered trade secrets and/or confidential.  Reproduction or
* distribution, in whole or in part, is forbidden except by express
* written permission of J.D. Edwards & Company.
*****************************************************************************
*/
```

Header Definition for a Business Function

Include the header definition for a business function. Use the source file name in the #ifndef statement. The source file and header file name should be the same.

Example: Including the Header Definition for a Business Function

The following example includes the header definition for a business function.

```
#ifndef __EXAMPLE_H
#define __EXAMPLE_H
```

Table Header Inclusions

The Table Header Inclusions section includes the table headers associated with tables directly accessed by this business function.

Use lower case in this section.

Example: Table Header Inclusions Section

The following example shows the Table Header Inclusion section:

```
/*****************************************************************************
**
* Table Header Inclusions
*****************************************************************************
/
#endif
```

External Business Function Header Inclusions

The External Business Function Header Inclusions section contains the business function headers associated with externally defined business functions that are directly accessed by this business function.

Use lower case in this section.

Example: External Business Function Header Inclusions Section

The following example shows the External Business Functions Header Inclusions section:

```
/*****************************************************************************
**
* External Business Function Header Inclusions
*****************************************************************************
/
#include <x1100.h>
```

Global Definitions

Use the Global Definitions section to define global constants used by the business function. Enter names in uppercase, separated by an underscore.

Example: Global Definitions Section

The following example defines global constants.

```
/*****************************************************************************
**
* Global Definitions
*****************************************************************************
/
#define EXPLANATION_LENGTH  31        /* Length of the explantion */
#define ACCOUNTS_PAYABLE    "AP"      /* Length of the company */
```

Note

This is not common practice by programmers. Multiple definitions may eventually arise. Global definition in .h remains in memory as long as J.D. Edwards software is running.

Structure Definitions

Define structures used by the business function in the Structure Definitions section. Structure names should be prefixed by the Source File Name to prevent conflicts with structures of the same name in other business functions.

Example: Structure Definitions Section

The following example defines structures used by the business function.

```
/*****************************************************************************
**
* Structure Definitions
*****************************************************************************
/
typedef struct tagDSB4100030ADDAMOUNT
{
 MATH_NUMERIC mnTotal;    /* Total */
 MATH_NUMERIC mnAmount;   /* Amount */
 short        nCounter;   /* Counter */
} DSB4100030ADDAMOUNT,     *LPDSB4100030ADDAMOUNT;
```

DS Template Type Definitions

Use the DS Template Type Definitions section to define the business functions contained in the source that corresponds to the header. The structure is generated through ER Data Structures. *Do not modify the DS Template Type Definitions section.*

Example: DS Template Type Definitions Section

The following example defines the data structure template types for the business function in the source:

```
/*****************************************************************************
**
 * DS Template Type Definitions

*****************************************************************************
/
/*************************************
```

```
 * TYPEDEF for Data Structure
 *    Template Name: Example For C Standards
 *    Template ID:   1234
 *    Generated:     Tue Sep 06 11:55:33 1994
 *
 * Do not edit the following typedef
 * To make modifications, use the Data Structure
 *      Tool to Generate a revised version, and paste from
 *      the clipboard.
 *
 * Note:  Copy the following line to the Comment Block
 *        preceding the C code for any functions referencing
 *        this data structure
 *
[Data Structure Template ID]
00001234
 *
 ************************************/
#ifndef DATASTRUCTURE_1234
#define DATASTRUCTURE_1234
typedef struct tagDS1234
{
  char             szDataField[30];             /* Data Field */
} DS1234, *LPDS1234;

#define                   IDERRszDataField                       4L
#endif
```

Source Preprocessing Definitions

The Source Preprocessing Definitions section defines the entry point of the business function and includes the opening bracket required by C functions.

Include all parameters on one line, if possible. The parameters should not run off the page. If necessary, stack the parameters in one column.

Example: Source Preprocessing Definitions Section

The following example shows the Source Preprocessing Definitions section:

```
/******************************************************************************
*
 * Source Preprocessing Definitions

******************************************************************************
/
#if defined (JDEBFRTN)
        #undef JDEBFRTN
#endif


#if defined (WIN32)
        #if defined (WIN32)
                #define JDEBFRTn(r) __declspec(dllexport) r
        #else
                #define JDEBFRTN(r) __declspec(dllimport) r
        #endif
#else
        #define JDEBFRTN(r) r
#endif


/******************************************************************************
*
 * Business Function Prototypes

******************************************************************************
/
JDEBFRTN (ID) JDEBFWINAPI GenericBusinessFunction

                   (LPBHVRCOM     lpBhvrCom,
                    LPVOID        lpVoid,
                    LPDSDXXXXXXXX lpDS);
```

Business Function Prototypes

Use the Business Function Prototype section to define business function prototypes for business functions in the source that corresponds to the header.

Example: Business Function Prototype Section

The following example defines a business function prototype:

```
/******************************************************************************
**
```

```
 * Business Function Prototype

****************************************************************************
/
ID ExampleForCStandards ( LPBHVRCOM   lpBhvrCom,
                          LPVOID      lpVoid,
                          LPDS1234    lpDS);
```

Internal Function Prototypes

This section contains a description and parameters of the function.

- Begin the function name with Ixxxxxxx_a, where:

  I = indicates an internal function

  xxxxxxx = the source file name

  a = the function name

- Ensure that the internal function corresponds to the header

Example: Internal Function Prototype Section

The following example defines internal function prototypes for an internal function in the source.

```
/****************************************************************************
**
 * Internal Function Prototype

****************************************************************************
/
ID Ixxxxxxxx_a ( LPBHVRCOM                    lpBhvrCom,
                 LPVOID                       lpVoid,
                 char                         szWorkCompany[6],
                 LPDSB4100030ADDAMOUNT        lpDSB4100030AddAmount);
```

Standard Source

The source file contains instructions for the business function.

Standard Source File

An example of a standard source file for a J.D. Edwards software business function appears on the following pages:

```
#include <jde.h>
#define bxxxxxxx_c



/*****************************************************************************
 *    Source File:  bxxxxxxx
 *
 *    Description:  Generic Business Function Source File
 *
 *       History:
 *          Date        Programmer  SAR# - Description
 *       ----------------------------------------------------------------------
 *   Author 06/06/1997                      - Created
 *
 * Copyright (c) J.D. Edwards World Source Company, 1996
 *
 * This unpublished material is proprietary to J.D.Edwards World Source
 * Company.
 * All rights reserved. The methods and techniques described herein are
 * considered trade secrets and/or confidential.  Reproduction or
 * distribution, in whole or in part, is forbidden except by express
 * written permission of J.D. Edwards World Source Company.

*****************************************************************************
*/
/*****************************************************************************
**
 * Notes:
 *

*****************************************************************************
*/

#include <bxxxxxxx.h>

/*****************************************************************************
**
 *  Business Function:  GenericBusinessFunction
```

```
 *
 *        Description:  Generic Business Function
 *
 *         Parameters:
 *           LPBHVRCOM           lpBhvrCom   Business Function
Communications
  *          LPVOID              lpVoid      Void Parameter - DO NOT USE!
 *           LPDSDXXXXXXX        lpDS        Parameter Data Structure
Pointer
 *

*****************************************************************************
*/
```

```
JDEBFRTN (ID) JDEBFWINAPI GenericBusinessFunction
                 (LPBHVRCOM      lpBhvrCom,
                  LPVOID         lpVoid,
                  LPDSDXXXXXXXX  lpDS)
{

/*****************************************************************************
**
    *  Variable declarations

*****************************************************************************
*/



/*****************************************************************************
*
    * Declare structures

*****************************************************************************
*/



/*****************************************************************************
**
    *  Declare pointers
```

```
*****************************************************************************
*/


/*****************************************************************************
**
    * Check for NULL pointers

*****************************************************************************
*/
   if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
       (lPVoid    == (LPVOID)    NULL) ||
       (lpDS      == (LPDSDXXXXXXXX) NULL))
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
      return ER_ERROR;
   }

/*****************************************************************************
**
    * Set pointers

*****************************************************************************
*/


/*****************************************************************************
**
    * Main Processing

*****************************************************************************
*/



/*****************************************************************************
**
    * Function Clean Up

*****************************************************************************
*/
```

```
   return (ER_SUCCESS);
}


/* Internal function comment block */
/******************************************************************************
**
 *   Function:  Ixxxxxxx_a // Replace "xxxxxxx" with source file number
 *                         // and "a" with the function name
 *      Notes:
 *
 *    Returns:
 *
 * Parameters:

******************************************************************************
*/
```

Source Sections

The source code may contain many different modules or sections. Sections are basically other source code files that contain various functions.

Business Function Name and Description

Use #include to define the name and description of the business function. Also use this section to maintain the modification log.

- For Source File: change example.c to your source member name.
- For Description: include a brief but clear explanation of what your business function accomplishes.
- Include a line with the date you first created the business function or checked out an existing business function and your initial and last name. Also include on the same line either the notation “Created” for a new business function or the SAR number for an enhancement or bug fix.

Example: Defining the Name and Description for a Business Function

The following example uses #include for defining the name and description of a business function:

```
#include <jde.h>
/******************************************************************************
 * Source File:  example.c
 *
 * Description:  Example of "c" member for coding standards.
 *        See bfstdsrc.c on the server in \b7\ins for a template.
```

```
 *
 *   History:                    Programmer                SAR# - Description
 *      Date
 *   ----------------------------------------------------------------------
 * Author  12/01/1994           MM247887                  Unknown - Created
****************************************************************************/
```

Copyright Notice

The Copyright section contains the J.D. Edwards & Company copyright notice and must be included in each source file. Do not make any changes to this section.

Example: Copyright Section

The following is an example of the standard copyright section:

```
/****************************************************************************
**
* Copyright (c) 1994
* J.D. Edwards & Company
 *
 * This unpublished material is proprietary to J.D. Edwards & Company.
 * All rights reserved.  The methods and techniques described herein are
 * considered trade secrets and/or confidential.  Reproduction or
 * distribution, in whole or in part, is forbidden except by express
 * written permission of J.D. Edwards & Company.

****************************************************************************
/
```

Notes

Use the Notes section to include information for anyone who may review the code in the future. For example, describe any peculiarities associated with the business function or any special logic.

Example: Notes Section

The following example shows a section for entering notes:

```
/****************************************************************************
 * Notes:
```

```
 ************************************************************************/
```

Header File for Associated Business Function

Include the header file associated with the business function using #include. Use the source file name in the #include statement. The source file and header file name should be the same.

Note

All other header files required by this business function will be included through this header file.

Example: Including the Header File Associated with the Business Function

The following example shows how to include the header file associated with this business function.

Change example.h to your source file name; the source file and header file names should be the same.

```
#include <example.h>
```

Business Function Header

The Business Function header section contains all of the information that is necessary to allow the Object Management Workbench to work correctly. Do not make any changes to this section.

Example: Business Function Header Section

The following example shows the Business Function header section:

```
/************************************************************************
*   Business Function:   FirstBusinessFunction
*
*      Description:   Business Function Number One
*
*      Parameters:
*       LPBHVRCOM              lpBvhrCom      Business Function Commications
*       LPVOID                 lpVoid         Void Parameter - DO NOT USE!
*       LPDS888                lpDS           Parameter Data Structure Pointer
************************************************************************/
```

Variable Declarations

The Variable Declarations section defines all required function variables.

- Define variables sequentially by type.
- Do not define more than one variable per line.
- Start your variable name with the Hungarian prefix so that we know at a glance the type.
- Use uppercase and lowercase to define variables.
- Be descriptive and, if possible, do not abbreviate.
- Initialize fields. Arrays must always be given a size.

Example: Variable Declarations Section

The following example shows the Variable Declarations section:

```
/*************************************************************************
 *  Variable declarations
 *************************************************************************/
 BOOL          bReturnValue                                    = FALSE;
 ID            idReturnCode                                    =
ER_SUCCESS;
 short         nCounter                                        = 0;
 char          cFlag                                           = ' ';
 char          szWorkCompany[6]                                =  "00000";
 char          szExplanation[(sizeof(EXPLANATION_LENGTH))]     = "\0";
 MATH_NUMERIC  mnAmount;
 JDEDATE       jdDate;
```

Declare Structures

Define any structures that are required by the function in this section. Align the type and name columns between the variable, structure, and pointer sections.

Example: Declare Structures Section

The following example shows how to declare structures:

```
/*************************************************************************
 * Declare structures
 *************************************************************************/
DSB4100030ADDAMOUNT    dsB4100030AddAmount;
```

```
DS1100                  ds1100;
F0011                   dsF0011;

KEY1_F0011              dsF0011Key;
```

Pointers

If any pointers are required by the function, define them here. Name the pointer so that it reflects the structure to which it is pointing. For example, lpDS1100 is pointing to the structure DS1100.

Example: Pointers Section

The following example shows the Pointers section:

```
/*************************************************************************
    * Pointers

*************************************************************************/
   LPX0051_DSTABLES lpdsTables = (LPX0051_DSTABLES) 0L;
```

Check for NULL Pointers

This section checks for parameter pointers that are NULL. If so, there is no reason to continue with the execution of this business function.

Example: Check for NULL Pointers Section

The following example checks for NULL pointers:

```
   /*************************************************************************
    * Check for NULL pointers
    *************************************************************************/
   if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
       (lpVoid    == (LPVOID)    NULL) ||
       (lpDS      == (LPDSDXXXXXXXX)  NULL))
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
      return ER_ERROR;
   }
```

Set Pointers

If you have defined pointers, you must assign values to the pointers. Use the Set Pointers section to assign values.

Example: Set Pointers Section

The following example assigns pointers values:

```
/*************************************************** ******************
 * Set pointers
 ************************************************************************/
  LPDSB4100030ADdAmount = &dsB4100030AddAmount;
  lpDS1100      = &ds1100;
```

Call Internal Function

Use the Call Internal Function section to call an internal function. Align all parameters in a list that does not run off the page.

Example: Call Internal Function Section

The following example shows how to format ‘C‘ code that calls an internal function:

```
        }
   }
/************************************************************************
 * Call Internal Function
 ************************************************************************/
 idReturnCode = (Ixxxxxxxx_a ( lpBhvrCom,     lpVoid,
                               szWorkCompany, lpDSB4100030AdDAmount) );
```

Internal Function Comment Block

This section contains a description and parameters of the function.

- Begin the function name with Ixxxxxxxx_a to indicate that it is an internal function, where:

  I = the designation for internal

  xxxxxxxx = the source file name

  a = the function name

- Identify each parameter with a number; and list the type, parameter name, and a short identifier within quotes. Use additional lines to briefly describe the parameter.

Example: Internal Function Description Section

The following is an Internal Function Description section:

```
   return ER_SUCCESS;
```

```
}
/* Internal function comment block*/
/*****************************************************************************
 *     Function:    Ixxxxxxxx_a    // Replace "xxxxxxxx" with source file
name
 *                                 // Replace "a" with the function name
 *       Note:
 *     Returns:
 *  Parameters:
 *****************************************************************************/
```

General Standards

This section contains general standards for J.D. Edwards C code.

Example: General Standards Section

The following example shows the general standards with which your 'C' code should comply:

```
}
/*****************************************************************************
 *    General Standards:
 *    . The coding emphasis is on clarity, not compactness.
 *    . Do not have tab characters in source.
 *    . As always, document your code with accurate and concise
 *      comments.
 *    . All documentation must bordered with '*'s as demonstrated
 *      in this example code.  The length of these lines should be
 *      the same as used here.  This will assure consistency
 *      between the different pc's.  Some monitors are using smaller
 *      fonts which allow more characters to be entered horizontally.
 *      For the same reason, do not allow your code to extend beyond
 *      this length.  Precede and follow each comment block by a blank
 *      line.  Do not use // for commenting.
 *    . All business functions should be written in ANSI C.
 *    . Avoid multiple returns.  If possible, the business function
 *      should return in only one place.
 *****************************************************************************/
```

Miscellaneous Source File Instructions

The preceding examples illustrate the various sections that are contained in the standard source file. The examples that follow provide additional information you must consider when programming business functions.

Using Braces

Place each opening or closing brace, { and }, on a separate line.

As each new block of code is started, ensure there are three spaces of indentation.

Always use braces around "if" statements, even when there is only one statement to execute.

Notice the pointer to the data structure is defined as lpDS. This is the standard default for the main business function structure.

Example: Use of Braces

The following example shows the standard for using braces:

```
/************************************************************************
    * Main Processing

************************************************************************/
   /** Check for Errors and Processing of Company Info **/
   if ( (idReturn == ER_SUCCESS) &&
        (lpDS->cProcessFlag == 'Y') )
   {
      if (lpDS->cValueFlag != '1')
      {
         ...
      }
      else
      {
         ...
      }

      /** Process All Period between the From and Thru Dates **/
      while (!dsInfo.bProcessed)
      {
         ...
      } /** end while **/
```

```
}
```

Declaring Variables and Structures and Checking for NULL Parameters in Internal Functions

Use the same format for internal functions as the main function to:

- Declare variables
- Declare structures
- Check for NULL parameters

Example: Declaring Variables and Structures and Checking for NULL Parameters for Internal Functions

The following example shows the section names to declare variables and structures and check for NULL parameters for an internal function:

```
/*****************************************************************************
 *  Variable declarations
 *****************************************************************************/
/*****************************************************************************
 * Declare structures
 *****************************************************************************/
/*****************************************************************************
 * Check for NULL pointer or NULL fields
 *****************************************************************************/
```

Calling an External Business Function

To call an external business function, you must use the data structure defined in the external business function.

Include in your header file that header file which contains the prototype and data structure definition of the external business function.

Example: Calling an External Business Function

The following example calls an external business function. Notice the pointer to the data structure is defined as lpDS1100. This is the standard for external business function structures. All calls to other business functions should have a return value. It is good practice to check the value of the return code.

```
/*---------------------------------------------------------------------
 *
```

```
    *   Determine if order type allows commitment to PA/PU ledger
    *  test for system code=40 and UDC=CT
    *
    *-------------------------------------------------------------------*/
   memset( (void *)(&dsUDC), (int)'\0', sizeof(DSD0005) );
   strcpy((char*)dsUDC.szSystemCode, (const char *)("40") );
   strcpy((char*)dsUDC.szRecordTypeCode, (const char *)("CT") );
   strcpy((char*)dsUDC.szUserDefinedCode,
          (const char *)(lpDS->szPODocumentType) );
   ParseNumericString( &dsUDC.mnKeyFieldLength, "2" ) ;
   dsUDC.cSuppressErrorMessage = '1';

   idReturn = jdeCallObject( "GetUDC", NULL, lpBhvrCom,
                  lpVoid, (void*)&dsUDC,
                  (CALLMAP*) NULL, (int) 0, (char*) NULL,
                  (char*) NULL, (int) 0 );

   if ( (!IsStringBlank( dsUDC.szErrorMessageId )) &&
        (idReturn == ER_SUCCESS))
   {
      lpDS->cErrorOnUpdate = 'Y';

      idReturn = ER_ERROR;
   }
```

Defining an Internal Function

To use an internal function, you should:

- Place all parameters on one line, if possible.
- Ensure the parameters do not run off the page.
- Stack the parameters in one column, if necessary.
- Ensure you return the proper status code where appropriate.

Example: Internal Function Definition

The following example shows the proper format of parameters when using an internal function:

```
ID Ixxxxxxxx_a  (LPBHVRCOM lpBhvrCom,     LPVOID            lpVoid,
               char szWorkCompany, LPDSB4100030AddAmount
lpDSB4100030AddAmount)
```

```
{
```

Terminating a Function

Always return a value with the termination of a function.

Example: Terminating a Function

The following example returns a value when the function is terminated:

```
MathAdd (&lpDSB4100030AddAmount->mnTotal,
         &lpDSB4100030AddAmount->mnAmount,
         &lpDSB4100030AddAmount->mnTotal);
lpDSB4100030ADdAMount->nCounter++;
 return ER_SUCCESS;
```

Changing the Standard Source C Code

You must note any code changes you make to the standard source for a business function. Include the following information:

- SAR - the SAR number
- Date - the date of the change
- Initials - the programmer's initials
- Comment - the reason for the change

Error Messages

Use J.D. Edwards messaging to get pertinent information to the end user in the most effective and user-friendly manner.

You can use simple messages or text substitution messages. Text substitution messages allow you to use variable text substitution. Substitution values are inserted into the message for the appropriate variable at runtime, which gives the user a customized message unique to every instance of the message.

There are two types of text substitution messages:

- Error messages (glossary group E)
- Workflow messages (glossary group Y)

For information on creating error messages or workflow messages, see *Messaging* in the *Tools Guide*.

Implementing Error Messages

The error handler processes error messages for business functions-based instructions included in the main processing section of the ‘C’ source code. This chapter details

instructions and provides examples for implementing error messages within a business function.

► Implementing error messages

From the Object Management Workbench, check out the business function.

1. Regenerate the data structure used by the business function passing the parameter.

Note
You do not need to change the structure. However, you must regenerate the function because new data is created automatically in the structure during the structure generation process.

The following example shows the new information that is created:

```
1.  /***************************************
2.   * TYPEDEF for Data Structure
3.   *    Template Name: Validate BUSINESS UNIT BEHVR
4.   *    Template ID:   36840
5.   *    Generated:     Sun Sep 11 12:24:37 1994
6.   *
7.   * Do not edit the following typedef
8.   * To make modifications, use the Data Structure
9.   *      Tool to Generate a revised version, and paste from
10.  *      the clipboard.
11.  *
12.  * Note:  Copy the following line to the Comment Block
13.  *        preceding the C code for any functions referencing
14.  *        this data structure
15.  *
16. [Data Structure Template ID]
17. 00036840
18.  *
19.  ************************************/
20. #ifndef DATASTRUCTURE_36840
21. #define DATASTRUCTURE_36840
22. typedef struct tagDS36840
23. {
24.   char             mnShortItemNumber;     /* Short Item Number */
25.   char             szDescription[31];     /* Description 01 */
```

```
26.   char             szErrorMessageID[5];   /* Error Message ID */
27.   char             cSuppressErrorMessage; /* Suppress Error Message */
28. } DS36840, FAR *LPDS36840;
29.
30. #define                   IDERRszShortItemNumber_1       1L
31. #define                   IDERRszDescription001_2        2L
32. #define                   IDERRszErrorMessageID_3        3L
33. #define                   IDERRcSuppressErrorMessage_4   4L
34.
35. #endif
```

The structure generation process now creates lines 26 and 27; they will be used in the call for error messaging.

2. Include the regenerated data structure in the .h file that is used by the business function.

   When you modify the .h file for the business function, ensure that the "#include mandata.h" statement has been replaced with the "#include jdedb.h" statement.

3. Open the .c file for the business function and locate all error points for which you will issue error messages. Typically, the error points are "return" statements. You must modify these statements to return one of the following three codes:

   - ER_SUCCESS - The business function completed normally, continue processing.
   - ER_WARNING - The business function detected a problem that is only a warning.
   - ER_ERROR - The business function detected an error and has issued a call to the error handling function.

   Structure jdeErrorSet as shown in the following example:

```
/*  Record not found  */
jdeErrorSet (lpBh vrCom, lpVoid, IDERRmnShortItemNumber, ”0052”);
```

The parameters for jdeErrorSet are defined as follows:

- lpBhvrCom - The standard Business function Communications structure.
- lpVoid - The parameter is type LPVOID and requires no value.
- IDERRmnShortItemNumber - Identifies the field in error. In this case, an error was detected for the Short Item Number field. The variable name is defined on line 25 of the structure that was defined on the previous page.
- "0052" - This is the message identifier number assigned to the message to be displayed. This number can be found by scanning through the Data Dictionary on the AS/400. Upon entering the Data Dictionary function, press F4 to search. When the next screen appears, enter a search key word, and specify a glossary

to of `E' for error messages (very important!). Search until you find an appropriate message, and use that assigned number here.

This message has an error level flag to which it is associated. If it is a hard error at the application level, then any process on the OK button and after will not execute.

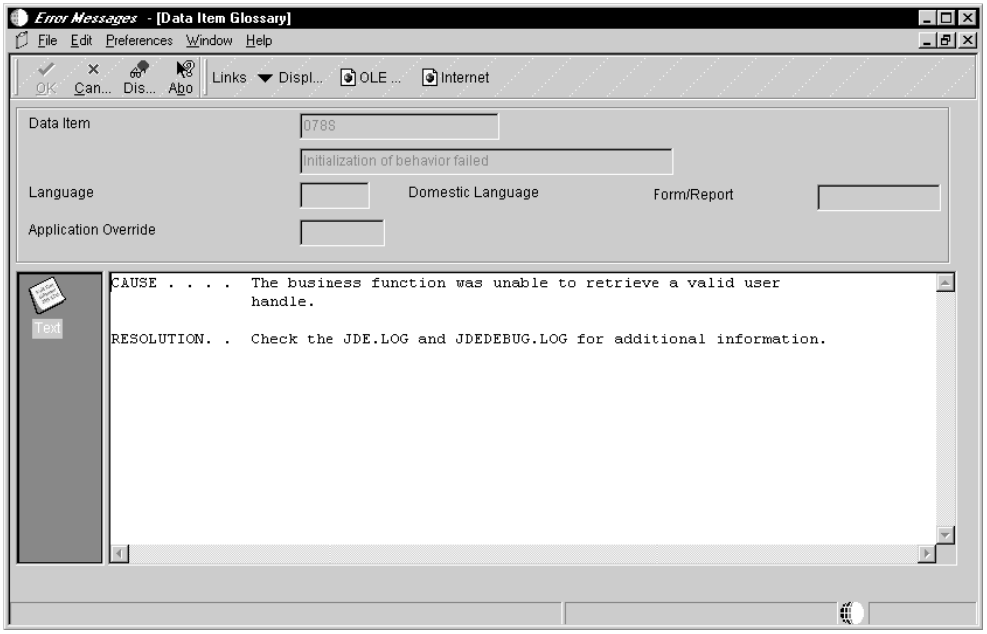
Example: Calling the Error Handler

```
   /************************************************************************
    * Main Processing

************************************************************************/


   memset( (void *) &dsUOMStruct, (int) '\0', sizeof(DSD4600040) ) ;
   strcpy( (char *) lpDS->szErrorMessageID, " " ) ;


   if ( (lpDS->cLotProcessType == '4' && !IsStringBlank(lpDS->szLotNumber))
||
        (lpDS->cLotProcessType == '5' || lpDS->cLotProcessType == '6' ||
         lpDS->cLotProcessType == '7' ) )
   {
      if ( jdestricmp( (char *) lpDS->szUOMLevel1,
                       (char *) lpDS->szPrimaryUOM ) != 0 )
      {
         idReturnValue = ER_ERROR ;
         strcpy( (char *) lpDS->szErrorMessageID, "0279" ) ;
         if ( lpDS->cSuppressErrorMsg != '1' )
         {
            jdeErrorSet ( lpBhvrCom, lpVoid, IDERRszUOMLevel1_22, "0279",
                          (void *) NULL ) ;
         }
      }
   }
```

Setting an Error for InitBehavior

When hUser fails, set the 078S error ID to "Initialization of Behavior Failed."

The following example shows the error message definition for 078S in the data dictionary:

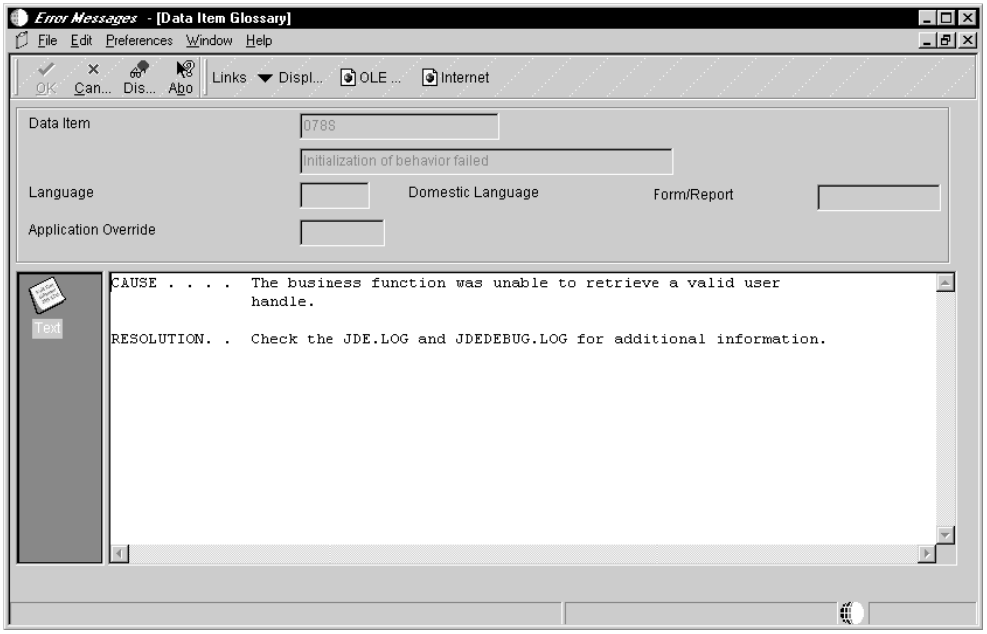


Using Text Substitution to Display Specific Error Messages

You can use the J.D. Edwards text substitution APIs for returning error messages within a business function. Text substitution is a flexible method for displaying a specific error message. This chapter briefly discusses how to implement the J.D. Edwards text substitution APIs in your business functions.

Text substitution is accomplished through the data dictionary. To use text substitution, you must first set up a data dictionary item that defines text substitution for your specific error message. A selection of error messages for JDB and JDE Cache have already been created and are listed in this chapter.

The following shows the definition for the JDB error message 078D - Open Table Failed in the data dictionary. For error message 078D, the specific table with which you are working is substituted for &1:



Error messages for cache and tables are critical in a configurable network computing (CNC) architecture. C programmers must set the appropriate error message when working with tables or cache APIs.

JDB API errors should substitute the name of the file against which the API failed. JDE cache API errors should substitute the name of the cache for which the API failed.

When calling errors that use text substitution, you must:

- Load a data structure with the information you want to substitute in the error message
- Call jdeErrorSet to set the error

The following example uses text substitution in JDB_OpenTable:

```
/**************************************************************************
*
 * Open the General Ledger Table F0911

**************************************************************************
/
eJDBReturn = JDB_OpenTable( hUser,
                            ID_F0911,
                            ID_F0911_DOC_TYPE__NUMBER___B,
                            idColF0911,
                            nNumColsF0911,
                            (char *)NULL,
                            &hRequestF0911);
```

```
if (eJDBReturn != JDEDB_PASSED)
{
   memset((void *)(&dsDE0022), (int)('\0'), sizeof(dsDE0022));
   strncpy(dsDE0022.szDescription, (const char *)("F0911"),
            sizeof(dsDE0022.szDescription));
   jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "078D", &dsDE0022);
}

/*************************************************************************
 * Check for NULL pointers

*************************************************************************/
if ((lpBhvrCom == )LPBHVRCOM)NULL) ||
    (lpVoid    == (LPVOID)NULL) ||
    (lpDS      == (LPDSD090049A)NULL))
{
   jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
   return ER_ERROR
}
```

DSDE0022 Data Structure

The data structure DSDE0022 contains the single item, szDescription[41]. Use the DSDE0022 data structure for JDB errors and JDE cache errors as the last parameter in jdeErrorSet.

DSDE0022 exists in jdeapper.h, which is included in the jdeapp.h header file. *You must include jdeapp.h in your business function header file.*

JDB Errors

Error ID	Description
078D	Open Table Failed
078E	Close Table Failed
078F	Insert to Table Failed
078G	Delete from Table Failed
078H	Update to Table Failed
078I	Fetch from Table Failed

078J	Select from Table Failed
078K	Set Sequence of Table Failed
078S *	Initialization of Behavior Failed

* 078S does not use text substitution

JDE Cache Errors

Error ID	Description
078L	Initialization of Cache Failed
078M	Open Cursor Failed
078N	Fetch from Cache Failed
078O	Add to Cache Failed
078P	Update to Cache Failed
078Q	Delete from Cache Failed
078R	Terminate of Cache Failed

Best Practices for 'C' Programming

This section contains tips and instructions for coding 'C' programs without errors or other unexpected runtime problems.

Copying Strings with strcpy vs. strncpy

When copying strings of the same length, such as business unit, the programmer may use the strcpy ANSI API. If the strings differ in length—as with a description—use the strncpy ANSI API with the size of minus one for the trailing NULL character.

```
/**************************************************************************
************
 * Variable Definitions

**************************************************************************
***********/
   char            szToBusinessUnit(13);
   char            szFromBusinessUnit(13);

   char            szToDescription(31);
```

```
   char           szFromDescription(41);

/**************************************************************************
************
 * Main Processing

**************************************************************************
***********/


   strcpy((char*) szToBusinessUnit,
          (const char*) szFromBusinessUnit );


   strncpy((char*) szToDescription,
           (const char*) szFromDescription,
            sizeof(szToDescription)-1 );
```

Using Memcpy to Assign JDEDATE Variables

When assigning JDEDATE variables, use the memcpy function. The memcpy function copies the information into the location of the pointer. A programmer could use a flat assignment, but may lose the scope of the local variable in the assignment. This could result in a lost data assignment.

```
/**************************************************************************
************
 * Variable Definitions

**************************************************************************
***********/
   JDEDATE      jdToDate;


/**************************************************************************
************
 * Main Processing

**************************************************************************
***********/
   memcpy((void*) &jdToDate,
           (void *) &lpDS->jdFromDate,
           sizeof(JDEDATE) );
```

Using MathCopy to Assign MATH_NUMERIC Variables

When assigning MATH_NUMERIC variables, use the MathCopy API. MathCopy copies the information, including Currency, into the location of the pointer. This API prevents any lost data in the assignment.

```
/*************************************************************************
**********
 * Variable Definitions

*************************************************************************
**********/
   MATH_NUMERIC      mnVariable;


/*************************************************************************
**********
 * Main Processing

*************************************************************************
**********/
   ZeroMathNumeric( &mnVariable );
   MathCopy( &mnVariable,
             &lpDS->Variable );
```

Initializing MATH_NUMERIC Variables

Initialize local MATH_NUMERIC variables with the ZeroMathNumeric API or memset all NULL values. If a MATH_NUMERIC is not initialized, invalid information, especially currency information, may be in the data structure. This can result in unexpected results at runtime.

```
/*************************************************************************
**********
 * Variable Definitions

*************************************************************************
**********/
   MATH_NUMERIC      mnVariable;


/*************************************************************************
**********
 * Main Processing

*************************************************************************
**********/
   ZeroMathNumeric( &mnVariable );
```

```
   MathCopy( &mnVariable,
             &lpDS->Variable );
```

Adding #include Statements

Always let the tool create includes for business functions and tables. When the programmer adds #include, capitalization issues may exist on other servers.

Copying Code and Data Structure Size

When copying code, use the correct size of the data structure. A programmer often copies code and changes the variable names, but fails to change the size of the data structure. The memset or strncpy may overwrite someplace in memory that is already used. This is often difficult to detect because unexpected results occur at runtime that are not always directly related to the business function.

Calling External Business Functions

The CallObject API should only call business functions that are defined in the Object Librarian. An internal business function should never be called by another business function. When code is copied from one source to another, change all the internal function names to match the current internal naming standard for the function. Internal functions from other business functions could be included from another source code, and must never be used outside the current source code in the Client/Server paradigm.

Mapping Data Structure Errors with jdeCallObject

Any Business Function calling another Business Function is required to use jdeCallObject. If jdeCallObject sets an Error on the data structure, the wrong field may highlight in an application because the Error Ids are not matched correctly with the next data structure.

The programmer needs to match the Ids from the original Business Function with the Error Ids with the Business Function in jdeCallObject. A data structure is used in the jdeCallObject to accomplish this task.

```
/************************************************************************
************

 * Variable declarations


************************************************************************
**********/

CALLMAP    cm_D0000026[2]    = {{IDERRmnDisplayExchgRate_62,
                                 IDERRmnExchangeRate_2}};

ID         idReturnCode      = ER_SUCCESS,      /* Return Code */
```

```
/**************************************************************************
************
 * Business Function structures

**************************************************************************
***********/
DSD0000026     dsD0000026    = {0};                                  /* Edit
Tolerance */


/**************************************************************************
************
 * Initializations

**************************************************************************
***********/
memset((void *)(&dsD0000026), (int)('\0'), sizeof(dsD0000026));


/**************************************************************************
************
 * Process - Exchange Rate (CRR)

**************************************************************************
***********/
if ((MathZeroTest(&lpDS->mnCurrencyRate))!=0)
{
   /* Edit amount for tolerance */
   memcpy((void *)(&dsD0000026.jdTransactionDate), (const void *)(&lpDS-
>jdGLDate),
           sizeof (dsD0000026.jdTransactionDate));
   MathCopy(&dsD0000026.mnExchangeRate, &lpDS->mnCurrencyRate);
   strncpy(dsD0000026.szCurrencyCodeFrom, (const char *)(lpDS-
>szTransactionCurrency),
           sizeof(dsD0000026.szCurrencyCodeFrom));
   strncpy(dsD0000026.szCurrencyCodeTo, (const char *)(lpDS-
>szBaseCoCurrency),
           sizeof(deD0000026.szCurrencyCodeTo));
   MathCopy(&dsD0000026.mnToleranceLimit, &lpdsI09UI002->potol);
  idReturnCode = jdeCallObject("EditExchangeRateTolerance", NULL,
                               lpBhvrCom, lpVoid,
                               (void *)&dsD0000026,
```

```
                                (CALLMAP *)&cm_D0000026, ND0000026,
                                (char *)NULL, (char *)NULL, (int)0);
}
```

Creating Data Dictionary Trigger Structures

When creating a structure for a Data Dictionary exit procedure, you **must** use the following format. The DD names follow the structure description. You **must** create a **new** structure for existing business functions. The items **must** be entered in the sequence shown or unpredictable results will occur. Namely, things will blow up. In the future, predefined structures will be provided.

```
B73.2 (3/98)
```

When creating the structure, use the following DD names:

```
idBhvrErrorId         = BHVRERRID
szBehaviorEditString = BHVREDTST
szDescription001      = DL01
```

The fourth item in the structure is the DD name of the field you are passing to your business function. In this case, it is Company.

Exception Handling

An exception is an error resulting from a problem or change in conditions that causes the microprocessor to stop what it is doing and handle the situation in a separate routine. Exception handling deals with exceptions as they arise during runtime.

An access violation is one kind of exception. Runtime and the UBE engine brackets all business function calls with exception handling. When a business function causes an exception that stops J.D. Edwards software, exception handling displays an access violation error message along with the business function that caused the error. The only way to debug the error when it occurs is to disable exception handling. Otherwise, exception handling clears the call stack and does not allow you to debug the function. To fix this so that you can debug the function when the access violation occurs, make the following changes in the jde.ini file:

In the [INTERACTIVE RUNTIME] section, change the EXCEPTION_Enabled flag to 0.

In the [UBE] section, add the flag Exception and set it to 0.