PeopleSoft

EnterpriseOne Xe Development Standards Business Function Programming PeopleBook

September 2000

J.D. Edwards World Source Company
7601 Technology Way
Denver, CO 80237

SKU XeEABS

Table of Contents

Business Function Programming Standards

A business function is an integral part of the OneWorld 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 OneWorld, it is important to establish firm rules regarding their makeup and usage.

Generally, there is not a standard method for creating C code that performs a specific action. 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 easier to maintain. The following sections are covered in this guide:

- ❑ Design Standards
- ❑ Standard Header and Source Files
- ❑ Error Messages
- ❑ Best Practices for C Programming

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, where each subroutine performs a discrete task.

Function Types

There are two types of functions with which you should be familiar:

- Business function or J.D. Edwards business function

- Function or internal function

Business Function

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

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

Application programmers spend most of their time writing J.D. Edwards business function 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. This section contains the following standards and guidelines:

- Naming Conventions
- Code Appearance Standards
- Business Function General Guidelines
- Portability
- J.D. Edwards Defined Structures

Naming Conventions

Naming standards ensure a consistent approach to identifying function objects as well as function sections. You should refer to this chapter when creating the following:

- Source and header file names
- Function names
- Variable names
- Business function data structure names
- Named Event Rule variable names

Source and Header File Names

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

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

Note: The source number is controlled by each of the application development groups. In the future, the source number will be automated and assigned by the Object Librarian. Until this tool is available, the source number is controlled by the application developers.

Function Names

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

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
id	unsigned long (identifier)
u	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

Examples: 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   wCalculationNumber;
unsigned long   idFunctionStatus;
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**

D = data structure

xxxx = the system code

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

A = If there are multiple data structures for a function, place an alphabetical character, such as A, B, C, and so on, at the end of the data structure name.

The data element in the data structure should use Hungarian Notation, with the data item alias appended. For example, if a data structure element is using LANO for the alias, the name would be mnSite_LANO.

See Also

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

Named Event Rule Variable Names

ER variables are named similar to C variables and should be formatted as follows: **xxx_yyzzzzzz_AAAA**

xxx = OneWorld 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

This chapter contains code appearance standards for the following:

- ❑ Declaring variables
- ❑ Using standard variables
- ❑ Using define statements
- ❑ Using typedef statements
- ❑ Creating function prototypes
- ❑ Indenting code
- ❑ Formatting compound statements
- ❑ Using function calls
- ❑ Including comments
- ❑ Using the function clean up area

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 also includes examples that show both standard and nonstandard formats.

Use the following check list when declaring variables:

- ❑ Declare variables using the following format:

```
datatype      variablename  =  initial value;     /* descriptive comment*/
```

- ❑ Declare all variables used within a business functions and internal function 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 there are multiple variables of the same type. 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 example on the following page.

- ❑ Use the naming convention 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* chapter.

- ❑ 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 when used.

- ❑ Always initialize all pointers to NULL, with an appropriate type call at the declaration line.

- ❑ Initialize all variables in the declaration, except for data structures.

- 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. This 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:

```
ID   MyBusinessFunction ( LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSNNNN lpDS)
{
ID   idReturn = BHVR_SUCCESS;   /* assume success */
F9860 dsF9860;
 DS1234  ds1234;
int i,nJDEDBReturn = JDEDB_PASSED;   /* return value from JDEDB API calls */
         / *   processing has already occurred, we're in the middle of the 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 flag used must be a Boolean type (BOOL).
- Name the flag variable to answer a question of TRUE or FALSE.

The following provides brief information for standard components that include:

- Flag Variables
- Input Parameters
- Fetch Variables

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 return 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, 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.
TableXXXXID	Where XXXX is the table name, such as F4101 and F41021. This is the variable used for defining the Table ID.
IndexXXXXID	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 example on the following page 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         nNumColFetch      = 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 upper case letters within the header file for the function. An example of this follows that 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 following 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 define statements. When using typedef statements, always name the object of the typedef statement using a descriptive, upper case 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 example that follows for using a typedef statement for a data structure.

Example: Using a 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 *Appendix C - 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 example below.
- ❑ 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 example below.
- ❑ Include a return type for every function. If a function does not return a value, use the key word "void" as the return type.
- ❑ Use the key word "void" in place of the parameter list if nothing is passed to the function.

Refer to the following examples of prototype definitions:

- Creating a business function prototype
- Creating an internal function prototype
- Creating a business function definition
- Creating an internal function definition

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 check list 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 confusing and may cause errors because braces are not used:

```
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 != zero. 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 Subsequents Modifications

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, "4363", (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 )
   {
      statement1;
      statement2;
   }
   statement3;
}
else if ( logical expression )
{
   statement4;
}
else
{
   statement5;
   statement6;
}
switch ( value )
{
   case CASE1:
      statement;
      statement:
      break;
   case CASE2:
      statement;
      statement;
      break;
   default:
      statement;
      break;
}
while ( logical expression )
{
   statement;
}
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 OneWorld functions, refer to the following check list 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 example below.
- ❑ 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.

The following examples are presented for using function calls:

- C Code that complies with the standard for calling a function
- C Code that violates the standard for calling a function
- Using jdeCallObject

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:

```
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 makes it easier for future maintenance and enhancement of the function.

Use the following check list 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.

Examples are presented for the following:

- Inserting comments in the source code
- Inserting comments in the header file

Example: Inserting Comments within the Source Code

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

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

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

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

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

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

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

 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 OneWorld 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. General guidelines are provided for the following topics:

- ❑ Initializing variables
- ❑ Typecasting
- ❑ Using constants
- ❑ Comparison testing
- ❑ Accessing the database efficiently
- ❑ Inserting function exit points
- ❑ Calling external business functions
- ❑ Using GENLNG as an address
- ❑ Allocating memory
- ❑ Releasing memory
- ❑ Initializing data structures for updating
- ❑ Using hRequest and hUser
- ❑ Inserting required parameters in lpDS for error messages

Initializing Variables

There are two types of variable initialization, 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 that 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;
HREUQEST         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) );
memcpy ((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. There are exceptions to this rule. 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 OneWorld 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 affect 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, where 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 there is only one result. For any other value, the function fails.

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

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


Creating True/False Test Comparison that Use Boolean Logic

Do not use embedded assignments for comparison tests. This is because embedded assignments are more difficult to read 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 >=, 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 is more readable than the prior example for embedding assignments:

```
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. There are several issues that must be considered whenever I/O to the J.D. Edwards database is performed.

When writing a new record to a data base 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 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 is 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 and 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.

There are no rules 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 be 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 ((lpBhvrCom == (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 ( (dsInfo.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 Bit

The address can be longer than 32 bit. If you pass the address for a GENLNG that exceeds 32 bit across the platform to a client that supports 32 bit, 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 array is maintained by the OneWorld tools. 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 successfull*/
   else
   {
      idReturnValue=ER_ERROR;
   }/*if memory allocation was unsuccessfull */
}
```

Retrieving address from 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 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 encounter 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 impacts 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 successfull*/
   else
   {
      idReturnValue=ER_ERROR;
   } /*if memory allocation was unsuccessfull */
}
```

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.

Insert Required Parameters in lpDS for Error Message

You must insert the parameters, cSuppressErrorMessage and szErrorMessageID in lpDS for an error message processing. The functionality for each are 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.

OneWorld business functions 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

Do *not* use the following method to create a C++ comment:

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

J.D. Edwards Defined Structures

There are two data types provided by OneWorld that you should be concerned with when you create business functions. They are 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 OneWorld 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 OneWorld. 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 OneWorld. 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. OneWorld Business Function Design generates the .c and .h template, including #include for tables and other business functions, and function, prototypes.

This section contains the following information:

- Standard header
- Standard source

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.

This chapter provides examples of the following:

- Standard source header
- Header sections

Standard Source Header

This chapter provides an example of standard header for a business function source file. An example of a standard source header for a OneWorld business function appears on the following page.

```
/******************************************************************************
 *     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. This chapter provides examples and a brief discussion for the following sections:

- Business function name and description
- Copyright notice
- Header definition for a business function
- Table header inclusions
- Global definitions
- Structure definitions
- DS template type definitions
- Source preprocessor definitions
- Business function prototype

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 to define 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 upper case, separated by an underscore.

Example: Global Definitions Section 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 OneWorld 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 correspond 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 OneWorld 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 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);
```

Business Function Prototypes

Use the Business Function Prototype section to define business function prototypes for business functions in the source, that correspond 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 Ixxxxxxxx_a, where:

  I = indicates an internal function

  xxxxxxxx = the source file name

  a = the function name

- Ensure that the internal function corresponds to the header

Example: Internal 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.

This chapter provides examples of the following:

- Standard source file
- Source sections
- Miscellaneous source file instructions

Standard Source File

An example of a standard source file for a OneWorld business function appears on the following two 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!
 *           LPDSDXXXXXXXX       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. This chapter provides examples and a brief description sections in standard source code.

- Business function name and description
- Copyright notice
- Notes
- Header file for associated business function
- Business function header
- Variable declarations
- Declare structures
- Pointers
- Check for NULL pointers
- Set pointers
- Call internal function
- Internal function comment block
- General standards

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 to define 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 name 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 Librarian 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

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 Upper and lower case to define variables.
- Be descriptive and do not abbreviate if possible.
- Initialize fields here. Arrays must always be given a size.

Example: Variable Declarations Section

The following example shows the Variable Declarations section:

```
/**************************************************************************
 *  Variable declarations
 **************************************************************************/
 BOOL          bReturnValue                                    = FALSE;         E
 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 the pointers values. 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 and 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 = designates internal

  xxxxxxxx = the source file name

  a = the function name

- Identify each parameter with a number; list the type, parameter name and a short identifier within quotes. Use additional lines to give a brief description for the parameter.

Example: Internal Function Description Section

The following is an Internal Function Description section:

```
    return ER_SUCCESS;
}
/* Internal function commet 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 then programming business functions. These examples include:

- Using braces
- Declaring variables and structures and checking for NULL parameters in internal functions
- Calling an external business function
- Defining an internal function
- Terminating a function

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 for 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 of the data structure defined in the external business function.

Include in your header file, the header file that 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 in 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. This 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 *OneWorld Tools Guide*.

This section describes the following:

❑ Implementing Error Messages

❑ Using Text Substitution to Display Specific Error Messages

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 Librarian

1. Check out the business function.
2. 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 OneWorld 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
```

Lines 26 and 27 are now created by the structure generation process, and will be used in the call for error messaging.

3. 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.

4. 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 following codes:

   - ER_SUCCESS - The business function completed normally, continue processing
   - ER_WARNING - The business function has detected a problem which is only a warning.
   - ER_ERROR - The business function has detected and 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.

The following code is one example for calling the error handler:

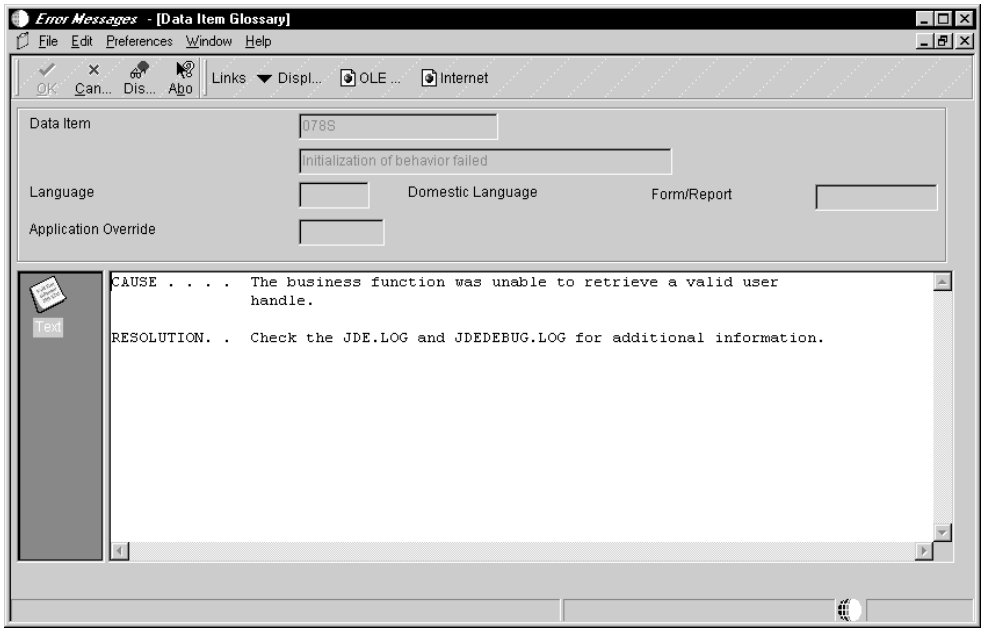
Example: Calling the Error Handler

```
/**************************************************************************
 * 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, “Initialization of Behavior Failed.

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

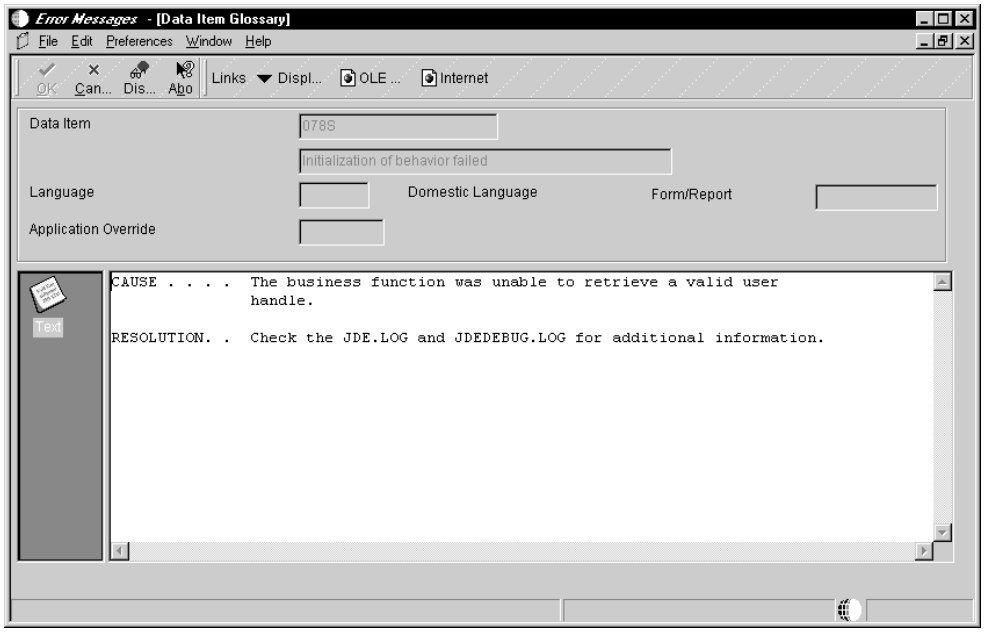


Using Text Substitution to Display Specific Error Messages

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 setup 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      == (LPDSD0900049A)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

Error ID	Description
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.

The following are presented:

- Copying strings with strcpt vs. strncpy
- Using memcpy to assign JDEDATE variables
- Using MathCopy to assign MATH_NUMERIC variables
- Initializing MATH_NUMERIC variables
- Adding #include statements
- Copying code and data structure size
- Calling external business functions
- Mapping data structures with jdeCallObject
- Creating data dictionary triggers structures
- Exception handling

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           szFromBuisnessUnit[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 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->mnVariable );
```

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 may be in the data structure, especially currency information. This can result in unexpected results at runtime.

```
/*****************************************************************************
 * Variable Definitions
 *****************************************************************************/
   MATH_NUMERIC        mnVariable;

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

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 OneWorld 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(dsD0000026.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 after 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 cause 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 OneWorld, 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.

Glossary

Glossary

AAI. See automatic accounting instruction.

action message. With OneWorld, users can receive messages (system-generated or user-generated) that have shortcuts to OneWorld forms, applications, and appropriate data. For example, if the general ledger post sends an action error message to a user, that user can access the journal entry (or entries) in error directly from the message. This is a central feature of the OneWorld workflow strategy. Action messages can originate either from OneWorld or from a third-party e-mail system.

activator. In the Solution Explorer, a parent task with sequentially-arranged child tasks that are automated with a director.

ActiveX. A computing technology, based on object linking and embedding, that enables Java applet-style functionality for Web browsers as well as other applications. (Java is limited to Web browsers at this time.) The ActiveX equivalent of a Java applet is an ActiveX control. These controls bring computational, communications, and data manipulation power to programs that can "contain" them. For example, certain Web browsers, Microsoft Office programs, and anything developed with Visual Basic or Visual C++.

advance. A change in the status of a project in the Object Management Workbench. When you advance a project, the status change might trigger other actions and conditions such as moving objects from one server to another or preventing check-out of project objects.

alphanumeric character. A combination of letters, numbers, and symbols used to represent data. Contrast with numeric character and special character.

API. See application programming interface.

APPL. See application.

applet. A small application, such as a utility program or a limited-function spreadsheet. It is generally associated with the programming language Java, and in this context refers to Internet-enabled applications that can be passed from a Web browser residing on a workstation.

application. In the computer industry, the same as an executable file. In OneWorld, an interactive or batch application is a DLL that contains programming for a set of related forms that can be run from a menu to perform a business task such as Accounts Payable and Sales Order Processing. Also known as system.

application developer. A programmer who develops OneWorld applications using the OneWorld toolset.

application programming interface (API). A software function call that can be made from a program to access functionality provided by another program.

application workspace. The area on a workstation display in which all related forms within an application appear.

audit trail. The detailed, verifiable history of a processed transaction. The history consists of the original documents, transaction entries, and posting of records, and usually concludes with a report.

automatic accounting instruction (AAI). A code that refers to an account in the chart of accounts. AAIs define rules for programs that automatically generate journal entries, including interfaces between Accounts Payable, Accounts Receivable, Financial Reporting, General Accounting systems. Each system that interfaces with the General Accounting system has AAIs. For example, AAIs can direct the General Ledger Post program to post a debit to a specific expense account and a credit to a specific accounts payable account.

batch header. The information that identifies and controls a batch of transactions or records.

batch job. A task or group of tasks you submit for processing that the system treats as a single unit during processing, for example, printing reports and purging files. The computer system

performs a batch job with little or no user interaction.

batch processing. A method by which the system selects jobs from the job queue, processes them, and sends output to the outqueue. Contrast with interactive processing.

batch server. A server on which OneWorld batch processing requests (also called UBEs) are run instead of on a client, an application server, or an enterprise server. A batch server typically does not contain a database nor does it run interactive applications.

batch type. A code assigned to a batch job that designates to which J.D. Edwards system the associated transactions pertain, thus controlling which records are selected for processing. For example, the Post General Journal program selects for posting only unposted transaction batches with a batch type of O.

batch-of-one immediate. A transaction method that allows a client application to perform work on a client workstation, then submit the work all at once to a server applicationfor further processing. As a batch process is running on the server, the client application can continue performing other tasks. See also direct connect, store and forward.

BDA. See Business View Design Aid.

binary string (BSTR). A length prefixed string used by OLE automation data manipulation functions. Binary Strings are wide, double-byte (Unicode) strings on 32-bit Windows platforms.

Boolean Logic Operand. In J.D. Edwards reporting programs, the parameter of the Relationship field. The Boolean logic operand instructs the system to compare certain records or parameters. Available options are:

EQ	Equal To.
LT	Less Than.
LE	Less Than or Equal To.
GT	Greater Than.
GE	Greater Than or Equal To.
NE	Not Equal To.
NL	Not Less Than.
NG	Not Greater Than.

browser. A client application that translates information sent by the World Wide Web. A client must use a browser to receive, manipulate, and display World Wide Web information on the desktop. Also known as a Web browser.

BSFN. See business function.

BSTR. See binary string.

BSVW. See business view.

business function. An encapsulated set of business rules and logic that can normally be reused by multiple applications. Business functions can execute a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the APIs that allow them to be called from a form, a database trigger, or a non-OneWorld application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.

business function event rule. See named event rule.

business view. Used by OneWorld applications to access data from database tables. A business view is a means for selecting specific columns from one or more tables whose data will be used in an application or report. It does not select specific rows and does not contain any physical data. It is strictly a view through which data can be handled.

Business View Design Aid (BDA). A OneWorld GUI ool for creating, modifying, copying, and printing business views. The tool uses a graphical user interface.

category code. In user defined codes, a temporary title for an undefined category. For example, if you are adding a code that designates different sales regions, you could change category code 4 to Sales Region, and define E (East), W (West), N (North), and S (South) as the valid codes. Sometimes referred to as reporting codes.

central objects. Objects that reside in a central location and consist of two parts: the central objects data source and central C components. The central objects data source contains OneWorld specifications, which are stored in a relational database. Central C components

contain business function source, header, object, library, and DLL files and are usually stored in directories on the deployment server. Together they make up central objects.

check-in location. The directory structure location for the package and its set of replicated objects. This is usually \\deploymentserver\release\path_code\package\ packagename. The sub-directories under this path are where the central C components (source, include, object, library, and DLL file) for business functions are stored.

child. See parent/child form.

client/server. A relationship between processes running on separate machines. The server process is a provider of software services. The client is a consumer of those services. In essence, client/server provides a clean separation of function based on the idea of service. A server can service many clients at the same time and regulate their access to shared resources. There is a many-to-one relationship between clients and a server, respectively. Clients always initiate the dialog by requesting a service. Servers passively wait for requests from clients.

CNC. See configurable network computing.

component. In the ActivEra Portal, an encapsulated object that appears inside a workspace. Portal components

configurable client engine. Allows user flexibility at the interface level. Users can easily move columns, set tabs for different data views, and size grids according to their needs. The configurable client engine also enables the incorporation of Web browsers in addition to the Windows 95- and Windows NT-based interfaces.

configurable network computing. An application architecture that allows interactive and batch applications, composed of a single code base, to run across a TCP/IP network of multiple server platforms and SQL databases. The applications consist of reusable business functions and associated data that can be configured across the network dynamically. The overall objective for businesses is to provide a future-proof environment that enables them to change organizational structures, business processes, and technologies independently of each other.

constants. Parameters or codes that you set and the system uses to standardize information processing by associated programs. Some examples of constants are: validating bills of material online and including fixed labor overhead in costing.

control. Any data entry point allowing the user to interact with an application. For example, check boxes, pull-down lists, hyper-buttons, entry fields, and similar features are controls.

core. The central and foundation systems of J.D. Edwards software, including General Accounting, Accounts Payable, Accounts Receivable, Address Book, Financial Reporting, Financial Modeling and Allocations, and Back Office.

CRP. Conference Room Pilot.

custom gridlines. A grid row that does not come from the database, for example, totals. To display a total in a grid, sum the values and insert a custom gridline to display the total. Use the system function Insert Grid Row Buffer to accomplish this.

data dictionary. The OneWorld method for storing and managing data item definitions and specifications. J.D. Edwards has an active data dictionary, which means it is accessed at runtime.

data mart. Department-level decision support databases. They usually draw their data from an enterprise data warehouse that serves as a source of consolidated and reconciled data from around the organization. Data marts can be either relational or multidimensional databases.

data replication. In a replicated environment, multiple copies of data are maintained on multiple machines. There must be a single source that "owns" the data. This ensures that the latest copy of data can be applied to a primary place and then replicated as appropriate. This is in contrast to a simple copying of data, where the copy is not maintained from a central location, but exists independently of the source.

data source. A specific instance of a database management system running on a computer. Data source management is accomplished

through Object Configuration Manager (OCM) and Object Map (OM).

data structure. A group of data items that can be used for passing information between objects, for example, between two forms, between forms and business functions, or between reports and business functions.

data warehouse. A database used for reconciling and consolidating data from multiple databases before it is distributed to data marts for department-level decision support queries and reports. The data warehouse is generally a large relational database residing on a dedicated server between operational databases and the data marts.

data warehousing. Essentially, data warehousing involves off-loading operational data sources to target databases that will be used exclusively for decision support (reports and queries). There are a range of decision support environments, including duplicated database, enhanced analysis databases, and enterprise data warehouses.

database. A continuously updated collection of all information a system uses and stores. Databases make it possible to create, store, index, and cross-reference information online.

database driver. Software that connects an application to a specific database management system.

database server. A server that stores data. A database server does not have OneWorld logic.

DCE. See distributed computing environment.

DD. See data dictionary.

default. A code, number, or parameter value that is assumed when none is specified.

detail. The specific pieces of information and data that make up a record or transaction. Contrast with summary.

detail area. A control that is found in OneWorld applications and functions similarly to a spreadsheet grid for viewing, adding, or updating many rows of data at one time.

direct connect. A transaction method in which a client application communicates interactively and directly with a server application. See also batch-of-one immediate, store and forward.

director. An interactive utility that guides a user through the steps of a process to complete a task.

distributed computing environment (DCE). A set of integrated software services that allows software running on multiple computers to perform in a manner that is seamless and transparent to the end-users. DCE provides security, directory, time, remote procedure calls, and files across computers running on a network.

DLL. See dynamic link library.

DS. See data structure.

DSTR. See data structure.

duplicated database. A decision support database that contains a straightforward copy of operational data. The advantages involve improved performance for both operational and reporting environments. See also enhanced analysis database, enterprise data warehouse.

dynamic link library (DLL). A set of program modules that are designed to be invoked from executable files when the executable files are run, without having to be linked to the executable files. They typically contain commonly used functions.

dynamic partitioning. The ability to dynamically distribute logic or data to multiple tiers in a client/server architecture.

embedded event rule. An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with business function event rule. See also event rule.

employee work center. This is a central location for sending and receiving all OneWorld messages (system and user generated) regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages. With respect to workflow, the Message Center is MAPI compliant and supports drag and drop work reassignment, escalation, forward and reply, and workflow monitoring. All messages from the message center can be viewed through OneWorld messages or Microsoft Exchange.

encapsulation. The ability to confine access to and manipulation of data within an object to the

procedures that contribute to the definition of that object.

enhanced analysis database. A database containing a subset of operational data. The data on the enhanced analysis database performs calculations and provides summary data to speed generation of reports and query response times. This solution is appropriate when external data must be added to source data, or when historical data is necessary for trend analysis or regulatory reporting. See also duplicated database, enterprise data warehouse.

enterprise data warehouse. A complex solution that involves data from many areas of the enterprise. This environment requires a large relational database (the data warehouse) that is a central repository of enterprise data, which is clean, reconciled, and consolidated. From this repository, data marts retrieve data to provide department-level decisions. See also duplicated database, enhanced analysis database.

enterprise server. A database server and logic server. See database server. Also referred to as host.

ER. See event rule.

ERP. See enterprise resource planning.

event. An action that occurs when an interactive or batch application is running. Example events are tabbing out of an edit control, clicking a push button, initializing a form, or performing a page break on a report. The GUI operating system uses miniprograms to manage user activities within a form. Additional logic can be attached to these miniprograms and used to give greater functionality to any event within a OneWorld application or report using event rules.

event rule. Used to create complex business logic without the difficult syntax that comes with many programming languages. These logic statements can be attached to applications or database events and are executed when the defined event occurs, such as entering a form, selecting a menu bar option, page breaking on a report, or selecting a record. An event rule can validate data, send a message to a user, call a business function, as well as many other actions. There are two types of event rules:

1 Embedded event rules.
2 Named event rules.

executable file. A computer program that can be run from the computer's operating system. Equivalent terms are "application" and "program.".

exit. 1) To interrupt or leave a computer program by pressing a specific key or a sequence of keys. 2) An option or function key displayed on a form that allows you to access another form.

facility. 1) A separate entity within a business for which you want to track costs. For example, a facility might be a warehouse location, job, project, work center, or branch/plant. Sometimes referred to as a business unit. 2) In Home Builder and ECS, a facility is a collection of computer language statements or programs that provide a specialized function throughout a system or throughout all integrated systems. For example, DREAM Writer and FASTR are facilities.

FDA. See Form Design Aid.

find/browse. A type of form used to:

1 Search, view, and select multiple records in a detail area.
2 Delete records.
3 Exit to another form.
4 Serve as an entry point for most applications.

firewall. A set of technologies that allows an enterprise to test, filter, and route all incoming messages. Firewalls are used to keep an enterprise secure.

fix/inspect. A type of form used to view, add, or modify existing records. A fix/inspect form has no detail area.

form. An element of OneWorld's graphical user interface that contains controls by which a user can interact with an application. Forms allow the user to input, select, and view information. A OneWorld application might contain multiple forms. In Microsoft Windows terminology, a form is known as a dialog box.

Form Design Aid (FDA). The OneWorld GUI development tool for building interactive applications and forms.

form interconnection. Allows one form to access and pass data to another form. Form interconnections can be attached to any event; however, they are normally used when a button is clicked.

form type. The following form types are available in OneWorld:

1 Find/browse.
2 Fix/inspect.
3 Header detail.
4 Headerless detail.
5 Message.
6 Parent/child.
7 Search/select.

fourth generation language (4GL). A programming language that focuses on what you need to do and then determines how to do it. Structured Query Language is an example of a 4GL.

graphical user interface (GUI). A computer interface that is graphically based as opposed to being character-based. An example of a character-based interface is that of the AS/400. An example of a GUI is Microsoft Windows. Graphically based interfaces allow pictures and other graphic images to be used in order to give people clues on how to operate the computer.

grid. See detail area.

GUI. See graphical user interface.

header. Information at the beginning of a table or form. This information is used to identify or provide control information for the group of records that follows.

header/detail. A type of form used to add, modify, or delete records from two different tables. The tables usually have a parent/child relationship.

headerless detail. A type of form used to work with multiple records in a detail area. The detail area is capable of of receiving input.

hidden selections. Menu selections you cannot see until you enter HS in a menu's Selection field. Although you cannot see these selections, they are available from any menu. They include such items as Display Submitted Jobs (33), Display User Job Queue (42), and Display User Print Queue (43). The Hidden Selections window displays three categories of selections: user tools, operator tools, and programmer tools.

host. In the centralized computer model, a large timesharing computer system that terminals communicate with and rely on for processing. In contrasts with client/server in that those users work at computers that perform much of their own processing and access servers that provide services such as file management, security, and printer management.

HTML. See hypertext markup language.

hypertext markup language. A markup language used to specify the logical structure of a document rather than the physical layout. Specifying logical structure makes any HTML document platform independent. You can view an HTML document on any desktop capable of supporting a browser. HTML can include active links to other HTML documents anywhere on the Internet or on intranet sites.

index. Represents both an ordering of values and a uniqueness of values that provide efficient access to data in rows of a table. An index is made up of one or more columns in the table.

inheritance. The ability of a class to recieve all or parts of the data and procedure definitions from a parent class. Inheritance enhances developement through the reuse of classes and their related code.

install system code. See system code.

integrated toolset. Unique to OneWorld is an industrial-strength toolset embedded in the already comprehensive business applications. This toolset is the same toolset used by J.D. Edwards to build OneWorld interactive and batch applications. Much more than a development environment, however, the OneWorld integrated toolset handles reporting and other batch processes, change management, and basic data warehousing facilities.

interactive processing. Processing actions that occur in response to commands you enter directly into the system. During interactive processing, you are in direct communication with the system, and it might prompt you for additional information while processing your request. See also online. Contrast with batch processing.

interface. A link between two or more computer systems that allows these systems to send information to and receive information from one another.

Internet. The worldwide constellation of servers, applications, and information available

to a desktop client through a phone line or other type of remote access.

interoperability. The ability of different computer systems, networks, operating systems, and applications to work together and share information.

intranet. A small version of the Internet usually confined to one company or organization. An intranet uses the functionality of the Internet and places it at the disposal of a single enterprise.

IP. A connection-less communication protocol that by itself provides a datagram service. Datagrams are self-contained packets of information that are forwarded by routers based on their address and the routing table information contained in the routers. Every node on a TCP/IP network requires an address that identifies both a network and a local host or node on the network. In most cases the network administrator sets up these addresses when installing new workstations. In some cases, however, it is possible for a workstation, when booting up, to query a server for a dynamically assigned address.

IServer Service. Developed by J.D. Edwards, this internet server service resides on the web server, and is used to speed up delivery of the Java class files from the database to the client.

ISO 9000. A series of standards established by the International Organization for Standardization, designed as a measure of product and service quality.

J.D. Edwards Database. See JDEBASE Database Middleware.

Java. An Internet executable language that, like C, is designed to be highly portable across platforms. This programming language was developed by Sun Microsystems. Applets, or Java applications, can be accessed from a web browser and executed at the client, provided that the operating system or browser is Java-enabled. (Java is often described as a scaled-down C++). Java applications are platform independent.

Java Database Connectivity (JDBC). The standard way to access Java databases, set by Sun Microsystems. This standard allows you to use any JDBC driver database.

JavaScript. A scripting language related to Java. Unlike Java, however, JavaScript is not an object-oriented language and it is not compiled.

jde.ini. J.D. Edwards file (or member for AS/400) that provides the runtime settings required for OneWorld initialization. Specific versions of the file/member must reside on every machine running OneWorld. This includes workstations and servers.

JDEBASE Database Middleware. J.D. Edwards proprietary database middleware package that provides two primary benefits:

1. Platform-independent APIs for multidatabase access. These APIs are used in two ways:
   a. By the interactive and batch engines to dynamically generate platform-specific SQL, depending on the datasource request.
   b. As open APIs for advanced C business function writing. These APIs are then used by the engines to dynamically generate platform-specific SQL.
2. Client-to-server and server-to-server database access. To accomplish this OneWorld is integrated with a variety of third-party database drivers, such as Client Access 400 and open database connectivity (ODBC).

JDECallObject. An application programming interface used by business functions to invoke other business functions.

JDENET. J.D. Edwards proprietary middleware software. JDENET is a messaging software package.

JDENET communications middleware. J.D. Edwards proprietary communications middleware package for OneWorld. It is a peer-to-peer, message-based, socket based, multiprocess communications middleware solution. It handles client-to-server and server-to-server communications for all OneWorld supported platforms.

job queue. A group of jobs waiting to be batch processed. See also batch processing.

just in time installation (JITI). OneWorld's method of dynamically replicating objects from the central object location to a workstation.

just in time replication (JITR). OneWorld's method of replicating data to individual

workstations. OneWorld replicates new records (inserts) only at the time the user needs the data. Changes, deletes, and updates must be replicated using Pull Replication.

KEY. A column or combination of columns that identify one or more records in a database table.

leading zeros. A series of zeros that certain facilities in J.D. Edwards systems place in front of a value you enter. This normally occurs when you enter a value that is smaller than the specified length of the field. For example, if you enter 4567 in a field that accommodates eight numbers, the facility places four zeros in front of the four numbers you enter. The result appears as: 00004567.

level of detail. 1) The degree of difficulty of a menu in J.D. Edwards software. The levels of detail for menus are as follows:

A	Major Product Directories.
B	Product Groups.
1	Basic Operations.
2	Intermediate Operations.
3	Advanced Operations.
4	Computer Operations.
5	Programmers.
6	Advanced Programmers Also known as menu levels.

2) The degree to which account information in the General Accounting system is summarized. The highest level of detail is 1 (least detailed) and the lowest level of detail is 9 (most detailed).

MAPI. See Messaging Application Programming Interface.

master table. A database table used to store data and information that is permanent and necessary to the system's operation. Master tables might contain data such as paid tax amounts, supplier names, addresses, employee information, and job information.

menu. A menu that displays numbered selections. Each of these selections represents a program or another menu. To access a selection from a menu, type the selection number and then press Enter.

menu levels. See level of detail.

menu masking. A security feature of J.D. Edwards systems that lets you prevent individual users from accessing specified menus or menu selections. The system does not display the menus or menu selections to unauthorized users.

Messaging Application Programming Interface (MAPI). An architecture that defines the components of a messaging system and how they behave. It also defines the interface between the messaging system and the components.

middleware. A general term that covers all the distributed software needed to support interactions between clients and servers. Think of it as the software that's in the middle of the client/server system or the "glue" that lets the client obtain a service from a server.

modal. A restrictive or limiting interaction created by a given condition of operation. Modal often describes a secondary window that restricts a user's interaction with other windows. A secondary window can be modal with respect to it's primary window or to the entire system. A modal dialog box must be closed by the user before the application continues.

mode. In reference to forms in OneWorld, mode has two meanings:

- An operational qualifier that governs how the form interacts with tables and business views. OneWorld form modes are: add, copy, and update.
- An arbitrary setting that aids in organizing form generation for different environments. For example, you might set forms generated for a Windows environment to mode 1 and forms generated for a Web environment to mode 2.

modeless. Not restricting or limiting interaction. Modeless often describes a secondary window that does not restrict a user's interaction with other windows. A modeless dialog box stays on the screen and is available for use at any time but also permits other user activities.

multitier architecture. A client/server architecture that allows multiple levels of processing. A tier defines the number of computers that can be used to complete some defined task.

named event rule. Encapsulated, reusable business logic created using through event rules rather than C programming. Contrast with embedded event rule. See also event rule.

NER. See named event rule.

network computer. As opposed to the personal computer, the network computer offers (in theory) lower cost of purchase and ownership and less complexity. Basically, it is a scaled-down PC (very little memory or disk space) that can be used to access network-based applications (Java applets, ActiveX controls) via a network browser.

network computing. Often referred to as the next phase of computing after client/server. While its exact definition remains obscure, it generally encompasses issues such as transparent access to computing resources, browser-style front-ends, platform independence, and other similar concepts.

next numbers. A feature you use to control the automatic numbering of such items as new G/L accounts, vouchers, and addresses. It lets you specify a numbering system and provides a method to increment numbers to reduce transposition and typing errors.

non–object librarian object. An object that is not managed by the object librarian.

numeric character. Digits 0 through 9 that are used to represent data. Contrast with alphanumeric characters.

object. A self-sufficient entity that contains data as well as the structures and functions used to manipulate the data. For OneWorld purposes, an object is a reusable entity that is based on software specifications created by the OneWorld toolset. See also object librarian.

object configuration manager (OCM). OneWorld's Object Request Broker and the control center for the runtime environment. It keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, the Object Configuration Manager directs access to it using defaults and overrides for a given environment and user.

object embedding. When an object is embedded in another document, an association is maintained between the object and the application that created it; however, any changes made to the object are also only kept in the compound document. See also object linking.

object librarian. A repository of all versions, applications, and business functions reusable in building applications. You access these objects with the Object Management Workbench.

object librarian object. An object managed by the object librarian.

object linking. When an object is linked to another document, a reference is created with the file the object is stored in, as well as with the application that created it. When the object is modified, either from the compound document or directly through the file it is saved in, the change is reflected in that application as well as anywhere it has been linked. See also object embedding.

object linking and embedding (OLE). A way to integrate objects from diverse applications, such as graphics, charts, spreadsheets, text, or an audio clip from a sound program. See also object embedding, object linking.

object management workbench (OMW). An application that provides check-out and check-in capabilities for developers, and aids in the creation, modification, and use of OneWorld Objects. The OMW supports multiple environments (such as production and development).

object-based technology (OBT). A technology that supports some of the main principles of object-oriented technology: classes, polymorphism, inheritance, or encapsulation.

object-oriented technology (OOT). Brings software development past procedural programming into a world of reusable programming that simplifies development of applications. Object orientation is based on the following principles: classes, polymorphism, inheritance, and encapsulation.

OCM. See object configuration manager.

ODBC. See open database connectivity.

OLE. See object linking and embedding.

OMW. Object Management Workbench.

OneWorld. A combined suite of comprehensive, mission-critical business applications and an embedded toolset for configuring those applications to unique business and technology requirements. OneWorld is built on the Configurable Network

Computing technology- J.D. Edwards' own application architecture, which extends client/server functionality to new levels of configurability, adaptability, and stability.

OneWorld application. Interactive or batch processes that execute the business functionality of OneWorld. They consist of reusable business functions and associated data that are platform independent and can be dynamically configured across a TCP/IP network.

OneWorld object. A reusable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects. See also object.

OneWorld process. Allows OneWorld clients and servers to handle processing requests and execute transactions. A client runs one process, and servers can have multiple instances. OneWorld processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes don't have to wait if the server is particularly busy.

OneWorld Web development computer. A standard OneWorld Windows developer computer with the additional components installed:

- JFC (0.5.1).
- Generator Package with Generator.Java and JDECOM.dll.
- R2 with interpretive and apllication controls/form.

online. Computer functions over which the system has continuous control. Users are online with the system when working with J.D. Edwards system provided forms.

open database connectivity (ODBC). Defines a standard interface for different technologies to process data between applications and different data sources. The ODBC interface is made up of a set of function calls, methods of connectivity, and representation of data types that define access to data sources.

open systems interconnection (OSI). The OSI model was developed by the International Standards Organization (ISO) in the early 1980s. It defines protocols and standards for the interconnection of computers and network equipment.

operand. See Boolean Logic Operand.

output. Information that the computer transfers from internal storage to an external device, such as a printer or a computer form.

output queue. See print queue.

package. OneWorld objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where on the deployment server the install program can find them. It is a point-in-time "snap shot" of the central objects on the deployment server.

package location. The directory structure location for the package and it's set of replicated objects. This is usually \\deployment server\release\path_code\package\ package name. The sub-directories under this path are where the replicated objects for the package will be placed. This is also referred to as where the package is built or stored.

parameter. A number, code, or character string you specify in association with a command or program. The computer uses parameters as additional input or to control the actions of the command or program.

parent/child form. A type of form that presents parent/child relationships in an application on one form. The left portion of the form presents a tree view that displays a visual representation of a parent/child relationship. The right portion of the form displays a detail area in browse mode. The detail area displays the records for the child item in the tree. The parent/child form supports drag and drop functionality.

partitioning. A technique for distributing data to local and remote sites to place data closer to the users who access. Portions of data can be copied to different database management systems.

path code. A pointer to a specific set of objects. A path code is used to locate:

1. Central Objects.
2. Replicated Objects.

platform independence. A benefit of open systems and Configurable Network Computing.

Applications that are composed of a single code base can be run across a TCP/IP network consisting of various server platforms and SQL databases.

polymorphism. A principle of object-oriented technology in which a single mnemonic name can be used to perform similar operations on software objects of different types.

portability. Allows the same application to run on different operating systems and hardware platforms.

portal. A configurable Web object that provides information and links to the Web. Portals can be used as home pages and are typically used in conjunction with a Web browser.

primary key. A column or combination of columns that uniquely identifies each row in a table.

print queue. A list of tables, such as reports, that you have submitted to be written to an output device, such as a printer. The computer spools the tables until it writes them. After the computer writes the table, the system removes the table identifier from the list.

processing option. A feature of the J.D. Edwards reporting system that allows you to supply parameters to direct the functions of a program. For example, processing options allow you to specify defaults for certain form displays, control the format in which information prints on reports, change how a form displays information, and enter beginning dates.

program temporary fix (PTF). A representation of changes to J.D. Edwards software that your organization receives on magnetic tapes or diskettes.

project. An Object Management Workbench object used to organize objects in development.

published table. Also called a "Master" table, this is the central copy to be replicated to other machines. Resides on the "Publisher" machine. the Data Replication Publisher Table (F98DRPUB) identifies all of the Published Tables and their associated Publishers in the enterprise.

publisher. The server that is responsible for the Published Table. The Data Replication Publisher Table (F98DRPUB) identifies all of the Published Tables and their associated Publishers in the enterprise.

pull replication. One of the OneWorld methods for replicating data to individual workstations. Such machines are set up as Pull Subscribers using OneWorld's data replication tools. The only time Pull Subscribers are notified of changes, updates, and deletions is when they request such information. The request is in the form of a message that is sent, usually at startup, from the Pull Subscriber to the server machine that stores the Data Replication Pending Change Notification table (F98DRPCN).

purge. The process of removing records or data from a system table.

QBE. See query by example.

query by example (QBE). Located at the top of a detail area, it is used to search for data to be displayed in the detail area.

redundancy. Storing exact copies of data in multiple databases.

regenerable. Source code for OneWorld business functions can be regenerated from specifications (business function names). Regeneration occurs whenever an application is recompiled, eitherfor a new platform or when new functionality is added.

relationship. Links tables together and facilitates joining business views for use in an application or report. Relationships are created based on indexes.

release/release update. A "release" contains major new functionality, and a "release update" contains an accumulation of fixes and performance enhancements, but no new functionality.

replicated object. A copy or replicated set of the central objects must reside on each client and server that run OneWorld. The path code indicates the directory the directory where these objects are located.

run. To cause the computer system to perform a routine, process a batch of transactions, or carry out computer program instructions.

SAR. See software action request.

scalability. Allows software, architecture, network, or hardware growth that will support software as it grows in size or resource

requirements. The ability to reach higher levels of performance by adding microprocessors.

search/select. A type of form used to search for a value and return it to the calling field.

selection. Found on J.D. Edwards menus, selections represent functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.

server. Provides the essential functions for furnishings services to network users (or clients) and provides management functions for network administrators. Some of these functions are storage of user programs and data and management functions for the file systems. It may not be possible for one server to support all users with the required services. Some examples of dedicated servers that handle specific tasks are backup and archive servers, application and database servers.

servlet. Servlets provide a Java-based solution used to address the problems currently associated with doing server-side programming, including inextensible scripting solutions. Servlets are objects that conform to a specific interface that can be plugged into a Java-based server. Servlets are to the server-side what applets are to the client-side.

software. The operating system and application programs that tell the computer how and what tasks to perform.

software action request (SAR). An entry in the AS/400 database used for requesting modifications to J.D. Edwards software.

special character. A symbol used to represent data. Some examples are *, &, #, and /. Contrast with alphanumeric character and numeric character.

specifications. A complete description of a OneWorld object. Each object has its own specification, or name, which is used to build applications.

Specs. See specifications.

spool. The function by which the system stores generated output to await printing and processing.

spooled table. A holding file for output data waiting to be printed or input data waiting to be processed.

SQL. See structured query language.

static text. Short, descriptive text that appears next to a control variable or field. When the variable or field is enabled, the static text is black; when the variable or field is disabled, the static text is gray.

store and forward. A transaction method that allows a client application to perform work and, at a later time, complete that work by connecting to a server application. This often involves uploading data residing on a client to a server.

structured query language (SQL). A fourth generation language used as an industry standard for relational database access. It can be used to create databases and to retrieve, add, modify, or deleta data from databases. SQL is not a complete programming language because it does not contain control flow logic.

subfile. See detail.

submit. See run.

subscriber. The server that is responsible for the replicated copy of a Published Table. Such servers are identified in the Subscriber Table.

subscriber table. The Subscriber Table (F98DRSUB), which is stored on the Publisher Server with the Data Replication Publisher Table (F98DRPUB) identifies all of the Subscriber machines for each Published Table.

subsystem job. Within OneWorld, subsystem jobs are batch processes that continually run independently of, but asynchronously with, OneWorld applications.

summary. The presentation of data or information in a cumulative or totaled manner in which most of the details have been removed. Many of the J.D. Edwards systems offer forms and reports that are summaries of the information stored in certain tables. Contrast with detail.

system. See application.

System Code. System codes are a numerical representation of J.D. Edwards and customer systems. For example, 01 is the system code for Address Book. System codes 55 through 59 are

reserved for customer development by customers. Use system codes to categorize within OneWorld. For example, when establishing user defined codes (UDCs), you must include the system code the best categorizes it. When naming objects such as applications, tables, and menus, the second and third characters in the object's name is the system code for that object. For example, G04 is the main menu for Acounts Payable, and 04 is its system code.

system function. A program module, provided by OneWorld, available to applications and reports for further processing.

table. A two-dimensional entity made up of rows and columns. All physical data in a database are stored in tables. A row in a table contains a record of related information. An example would be a record in an Employee table containing the Name, Address, Phone Number, Age, and Salary of an employee. Name is an example of a column in the employee table.

table design aid (TDA). A OneWorld GUI tool for creating, modifying, copying, and printing database tables.

table event rules. Use table event rules to attach database triggers (or programs) that automatically run whenever an action occurs against the table. An action against a table is referred to as an event. When you create a OneWorld database trigger, you must first determine which event will activate the trigger. Then, use Event Rules Design to create the trigger. Although OneWorld allows event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

TAM. Table Access Management.

TBLE. See table.

TC. Table conversion.

TCP/IP. Transmission Control Protocol/Internet Protocol. The original TCP protocol was developed as a way to interconnect networks using many different types of transmission methods. TCP provides a way to establish a connection between end systems for the reliable delivery of messages and data.

TCP/IP services port. Used by a particular server application to provide whatever service the server is designed to provide. The port number must be readily known so that an application programmer can request it by name.

TDA. See table design aid.

TER. See table event rules.

Terminal Identification. The workstation ID number.Terminal number of a specific terminal or IBM user ID of a particular person for whom this is a valid profile.Header Field: Use the Skip to Terminal/User ID field in the upper portion of the form as an inquiry field in which you can enter the number of a terminal or the IBM user ID of a specific person whose profile you want the system to display at the top of the list. When you first access this form, the system automatically enters the user ID of the person signed on to the system.Detail Field: The Terminal/User ID field in the lower portion of the form contains the user ID of the person whose profile appears on the same line.A code identifying the user or terminal for which you accessed this window.

third generation language (3GL). A programming language that requires detailed information about how to complete a task. Examples of 3GLs are COBOL, C, Pascal and FORTRAN.

token. A referent to an object used to determine ownership of that object and to prevent non-owners from checking the object out in Object Management Workbench. An object holds its own token until the object is checked out, at which time the object passes its token to the project in which the object is placed.

trigger. Allow you to attach default processing to a data item in the data dictionary. When that data item is used on an application or report, the trigger is invoked by an event associated with the data item. OneWorld also has three visual assist triggers: calculator, calendar and search form.

UBE. Universal batch engine.

UDC Edit Control. Use a User-Defined Code (UDC) Edit Control for a field that accepts only specific values defined in a UDC table. Associate a UDC edit control with a database item or dictionary item. The visual assist Flashlight automatically appears adjacent to the UDC edit

control field. When you click on the visual assist Flashlight, the attached search and select form displays valid values for the field. To create a UDC Edit Control, you must:

- Associate the data item with a specific UDC table in the Data Dictionary.
- Create a search and select form for displaying valid values from the UDC table.

uniform resource identifier (URI). A character string that references an internet object by name or location. A URL is a type of URI.

uniform resource locator (URL). Names the address (location) of a document on the Internet or an intranet. A URL includes the document's protocol and server name. The path to the document might be included as well. The following is an example of a URL: http://www.jdedwards.com. This is J.D. Edwards Internet address.

URI. See uniform resource identifier.

URL. See uniform resource locator.

user defined code (type). The identifier for a table of codes with a meaning you define for the system, such as ST for the Search Type codes table in Address Book. J.D. Edwards systems provide a number of these tables and allow you to create and define tables of your own. User defined codes were formerly known as descriptive titles.

user defined codes (UDC). Codes within software that users can define, relate to code descriptions, and assign valid values. Sometimes user defined codes are referred to as a generic code table. Examples of such codes are unit-of-measure codes, state names, and employee type codes.

UTB. Universal Table Browser.

valid codes. The allowed codes, amounts, or types of data that you can enter in a field. The system verifies the information you enter against the list of valid codes.

visual assist. Forms that can be invoked from a control to assist the user in determining what data belongs in the control.

vocabulary overrides. A feature you can use to override field, row, or column title text on forms and reports.

wchar_t. Internal type of a wide character. Used for writing portable programs for international markets.

web client. Any workstation that contains an internet browser. The web client communicates with the web server for OneWorld data.

web server. Any workstation that contains the IServer service, SQL server, Java menus and applications, and Internet middleware. The web server receives data from the web client, and passes the request to the enterprise server. When the enterprise server processes the information, it sends it back to the web server, and the web server sends it back to the web client.

WF. See workflow.

window. See form.

workflow. According to the Workflow Management Coalition, worlflow means "the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.".

workgroup server. A remote database server usually containing subsets of data replicated from a master database server. This server does not performance an application or batch processing. It may or may not have OneWorld running (in order to replicate data).

workspace. In the ActivEra Portal, the main section of the Portal. A user might have access to several workspaces, each one configured differently and containing its own components.

worldwide web. A part of the Internet that can transmit text, graphics, audio, and video. The World Wide Web allows clients to launch local or remote applications.

z file. For store and forward (network disconnected) user, OneWorld store and forward applications perform edits on static data and other critical information that must be valid to process an order. After the initial edits are complete, OneWorld stores the transactions in work tables on the workstation. These work table are called Z files. When a network connection is established, Z files are uploaded to the enterprise server and the transactions are

edited again by a master business function. The master business function will then update the records in your transaction files.

Index

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

Z