PeopleSoft

EnterpriseOne 8.9
開発スタンダード：ビジネス関数プログラミング
PeopleBook

2003 年 9 月

PeopleSoft EnterpriseOne 8.9
開発スタンダード:ビジネス関数プログラミング PeopleBook
SKU AC89JBS0309

目次

ビジネス関数プログラミング標準

ビジネス関数は、J.D. Edwards ソフトウェア・ツール・セットの統合的な部品です。アプリケーション開発者はビジネス関数を使用することで、アプリケーションやバッチ処理イベントにカスタム関数を関連付けることができます。

特定のアクションを実行する C 言語コードの作成には、標準的な方法はありません。C 言語コードは、コードをプログラムし、コンパイルしたプログラマに依存するものです。本書では、効率的かつ管理しやすいビジネス関数のコーディング標準について説明します。

プログラム・フロー

C 言語関数のプログラム・フローでは、すべてのコード・セグメントをモジュール化する必要があります。読みやすさと管理のために、コードを論理的な単位に分割します。

ビジネス関数はその一部が J.D. Edwards ソフトウェア・ツールを使用して作成されているため、J.D. Edwards ビジネス関数ともいいます。ビジネス関数は、他のアプリケーションやビジネス関数で使用できるようにチェックインおよびチェックアウトされます。

内部関数を使用できるのは、ソース・ファイル内のみです。他のソース・ファイルからは内部関数にアクセスできません。内部関数は、ビジネス関数が呼び出す共通ルーチンのモジュールを意図して作成されています。

命名規則

標準化された命名規則により、関数オブジェクトおよび関数セクションを識別するための一貫性のあるアプローチが実現します。

ソース・ファイル名とヘッダー・ファイル名

ソース・ファイル名とヘッダー・ファイル名は 8 文字以内で、次の形式で指定する必要があります。

bxxyyyyy

各変数は次の意味を表します。

b = ソース・ファイルまたはヘッダー・ファイル

xx（第 2 桁と第 3 桁） = システム・コード。たとえば、

01 – 住所録

04 – 買掛管理

yyyyy（末尾 5 桁） = システム・コードの連番。たとえば、

00001 – システム・コードの最初のソース・ファイルまたはヘッダー・ファイル

00002 – システム・コードの 2 番目のソース・ファイルまたはヘッダー・ファイル

C ソース・ファイル名および関連ヘッダー・ファイル名は、同じにする必要があります。

次の表に、この命名規則の例を示します。

システム	システム・コード	ソース番号	ソース・ファイル	ヘッダー・ファイル
住所録	01	10	b0100010.c	b0100010.h
売掛管理	04	58	b0400058.c	b0400058.h
一般会計	09	2457	b0902457.c	b0902457.h

関数名

内部関数は 42 文字以内で、次の形式で指定する必要があります。

Ixxxxxxx_a

各変数は次の意味を表します。

I = 内部関数

xxxxxxx = ソース・ファイル名

a = 関数の記述。関数記述の長さは 32 文字以内です。できるだけわかりやすい記述とし、ValidateTransactionCurrencyCode のように各単語の最初の文字は大文字とします。可能な場合は、関数の主要テーブル名または関数の目的を表すようにします。

次に関数名の例を示します。

I4100040_CompareDate

注:

I に続けてアンダースコアを使用しないでください。

変数名

変数はプログラム内の値の保管場所であり、数値および文字列を保管できます。変数はコンピュータのメモリに保管されます。変数は char や MATH_NUMERIC のようなキーワードおよび関数と共に使用され、プログラムの始まりで宣言する必要があります。

変数名の長さは 32 文字以内です。できるだけわかりやすい記述とし、各単語の最初の文字は大文字とします。

次の表に示すように、すべての変数名にハンガリアン接頭辞表記法を使用する必要があります。

プレフィックス（接頭辞）	**説明**
c	JCHAR
sz	NULL 終了 JCHAR 文字列
z	ZCHAR
zz	NULL 終了 ZCHAR 文字列

n	short 型
l	long 型
b	ブール型
mn	MATHNUMERIC
jd	JDEDATE
lp	long 型ポインタ
i	integer 型
by	バイト
ul	符号なし long 型（識別子）
us	符号なし Short 型
ds	データ構造体
h	ハンドル
e	列挙型
id	id long integer 型、戻り値に使用される J.D. Edwards データ型

例：変数名のハンガリアン表記法

次の変数名はハンガリアン表記法を使用しています。

```
JCHAR                   cPaymentRecieved;
JCHAR                   szCompanyNumber = _J("00000");
short                   nLoopCounter;
long int                lTaxConstant;
BOOL                    bIsDateValid;
MATH_NUMERIC            mnAddressNumber;
JDEDATE                 jdGLDate;
LPMATH_NUMERIC          lpAddressNumber;
int                     iCounter;
```

byte	byOffsetValue;
unsigned long	ulFunctionStatus;
D0500575A	dsInputParameters;
JDEDB_RESULT	idJDEDBResult;

ビジネス関数データ構造体名

ビジネス関数イベント・ルールおよびビジネス関数のデータ構造体は、次の形式で指定します。

DxxyyyyA

各変数は次の意味を表します。

D = データ構造体

xx（第 2 桁と第 3 桁） = システム・コード。たとえば、

01 – 住所録

04 – 買掛管理

yyyy = 次の番号（採番割当ては、各自のアプリケーション・グループの現在のプロシージャに従います。）

A = 関数に複数のデータ構造体があることを示すためにデータ構造体名の末尾に追加される、A、B、C などのアルファベット文字。関数にデータ構造体が 1 つしかない場合も、名称には A を含める必要があります。

次にビジネス関数データ構造体名の例を示します。

D050575A

参照

- 『開発ツール』ガイドの「ビジネス関数データ構造体の作成」

読みやすさ

コードが読みやすいほど、デバッグと管理が容易になります。ここでは、変更ログの管理、コメントの挿入、コードのインデント設定、および複合ステートメントのフォーマット設定により、さらに読みやすいコードにするためのガイドラインを説明します。

ソースおよびヘッダー・コードの変更ログの管理

ビジネス関数の標準ソースおよびヘッダーに対して行うコード変更は、記録しておく必要があります。これには次の情報が含まれます。

- SAR – SAR 番号
- 日付 – 変更日
- イニシャル – プログラマのイニシャル
- コメント – 変更理由

コメントの挿入

ビジネス関数の目的と意図したアプローチを記述するコメントを挿入します。コメントを使用すると、関数の将来の管理と機能拡張が容易になります。

コメントの挿入時には、次のチェックリストを使用してください。

- 必ず/*comment */の形式にします。//形式のコメントを使用すると移植できなくなります。
- コメントは、そこで記述するステートメントの直後に同じ配置で挿入します。
- コメントは長さ 80 文字以内とします。

例:コメントの挿入

次の例に、コードにコメント・ブロックとインライン・コメントを挿入する際の適切な方法を示します。

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

if ( statement )
{
   statements
} /* inline comments indicate the 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
 *************************************************************/
```

コードのインデント設定

コード・ブロック内で実行されるステートメントは、そのブロック内でインデントする必要があります。標準インデントはスペース 3 個分です。

注:

使用するエディタの環境を、タブの文字数が 3 で Tab 文字がオフになるように設定します。これにより、Tab キーを押すたびに、Tab 文字ではなくスペース 3 個が挿入されます。自動インデント機能はオンにしてください。

例:コードのインデント設定

次にコードの標準的なインデント方法を示します。

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

```
}
```

複合ステートメントのフォーマット設定

複合ステートメントとは、中カッコで囲まれた 1 つまたは複数のステートメントが後に続くステートメントです。関数ブロックは、複合ステートメントのわかりやすい一例です。制御ステートメント（while、for）や選択ステートメント（if、switch）も、複合ステートメントの一例です。

一般的な C 言語のコーディングでは、制御ステートメントまたは選択ステートメントにステートメントが 1 つしか続かない場合は、中カッコが省略されています。ただし、次の理由により、すべての複合ステートメントに中カッコを使用する必要があります。

- 中カッコを省略すると、エラーの原因となることがあります。
- 中カッコにより、すべての複合ステートメントが同じ方法で確実に処理されます。
- ネストした複合ステートメントの場合は、中カッコを使用すると、特定のコード・ブロックに属するステートメントが明確になります。
- 中カッコにより、その後の修正がさらに容易になります。

複合ステートメントのフォーマット設定時には、次のガイドラインに従ってください。

- 複合ステートメントでは、常に 1 行に 1 ステートメントとします。
- 制御ステートメントまたは選択ステートメントに続くステートメントを入れるには、常に中カッコを使用します。
- 中カッコは、最初の制御ステートメントまたは選択ステートメントと位置揃えする必要があります。
- 制御ステートメントまたは選択ステートメントで評価される論理計算式が 1 行に収まらない場合は、複数行に分ける必要があります。複数の論理計算式を分ける場合は、新しい行を論理演算子で始めないでください。論理演算子は前の行に残しておく必要があります。
- 複数の論理計算式を評価する場合は、カッコを使用して優先順位を明示します。
- 関数ブロックを除き、複合ステートメントでは変数を宣言しないでください。
- すべての複合ステートメントに中カッコを使用します。
- 左右の中カッコ{または}は、別々の行に置きます。

例：複合ステートメントのフォーマット設定

次の例に、使用しやすく誤りを防ぐ複合ステートメントのフォーマット方法を示します。

```
/*
 * Do the Issues Edit Line if the process edits is either
 * blank or set to SKIP_COMPLETIONS. The process edits is
 * set to SKIP_COMPLETIONS if Hours and Quantities is in
 * interactive mode and Completions is Blind in P31123.
 */
```

```
if ((dsWorkCache.PO_cIssuesBlindExecution == _J('1')) &&
    ((dsCache.cPayPointCode == _J('M'))            ||
     (dsCache.cPayPointCode == _J('B')))             &&
    (lpDS->cProcessEdits != ONLY_COMPLETIONS))
{
   /* Process the Pay Point line for Material Issues */
   idReturnCode = I3101060_BlindIssuesEditLine(&dsInternal,
                                               &dsCache,
                                               &dsWorkCache);
}
```

例:フローを明確にするために中カッコを使用した場合

次の例に、フローを明確にして誤りを防ぐために中カッコを使用した場合のコードを示します。

```
if(idJDBReturn != JDEDB_PASSED)
{
   /* If not add mode, record must exist */
   if ((lpdsInternal->cActionCode != ADD_MODE) &&
       (lpdsInternal->cActionCode != ATTACH_MODE))
   {
      /* Issue Error 0002 - Work Order number invalid */
      jdeStrncpy((JCHAR*)(lpdsInternal->szErrorMessageID),
                 (const JCHAR*)_J("0002"),
                 DIM(lpdsInternal->szErrorMessageID)-1);
      lpdsInternal->idFieldID = IDERRmnOrderNumber_15;
      idReturnCode = ER_ERROR;
   }
}
else
{
   /* If in add mode and the record exists, issue error and exit */
   if (lpdsInternal->cActionCode == ADD_MODE)
   {
      /* Issue Error 0002 - Work Order number invalid */
      jdeStrncpy((JCHAR*)(lpdsInternal->szErrorMessageID),
                 (const JCHAR*)_J("0002"),
                 DIM(lpdsInternal->szErrorMessageID)-1);
```

```
      lpdsInternal->idFieldID = IDERRmnOrderNumber_15;
      idReturnCode = ER_ERROR;
   }
   else
   {
      /*
       * Set flag used in determining if the F4801 record should be sent
       * in to the modules
       */
       lpdsInternal->cF4801Retrieved = _J('1');
    }
}
```

例：その後の修正が容易になるように中カッコを使用した場合

中カッコを使用すると、後でコードを修正する際に誤りを防ぐことができます。次のような場合があります。元のコードには、行数が事前定義済みの上限に達していないかどうかを調べるテストが含まれます。行数が上限を超えると戻り値に特定の値が割り当てられます。その後、特定の戻り値を割り当てる他に、エラー・メッセージを発行するように決定したとします。これは、行数が上限を超える場合にのみ、両方のステートメントを実行するためです。しかし、これでは idReturn は nLines の値に関係なく ER_ERROR に設定されてしまいます。最初から中カッコを使用していれば、このような誤りは避けられたはずです。

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


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


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


STANDARD MODIFIED
```

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

例:複数の論理計算式の処理

次の例に、複数の論理計算式の処理方法を示します。

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


{
   statements
}
```

変数とデータ構造体の宣言と初期化

変数とデータ構造体は、データの保管に使用する前に定義して初期化する必要があります。ここでは、両者を宣言して初期化する方法について説明します。define ステートメントの使用、typedef の使用、関数プロトタイプの作成、変数の初期化、データ構造体の初期化、および標準変数の使用について、個別に説明します。

define ステートメントの使用

define ステートメントは、プログラムの始まりで定数を設定する指示です。define ステートメントは常にシャープ記号(#)で始まります。すべてのビジネス関数はシステム・ヘッダー・ファイル jde.h にあります。システム全体の define ステートメントは、システム・ヘッダー・ファイルに組み込まれています。

特定の関数に define ステートメントが必要な場合は、できるだけその関数のソース・ファイルに define ステートメントを大文字で組み込みます。ステートメントの直後に、ヘッダー・ファイル・インクルード・ステートメントを指定する必要があります。

通常、define ステートメントはヘッダー・ファイルではなくソース・ファイルに置いてください。ヘッダー・ファイルに置くと、同じ定数を異なる値で再定義した場合に、予測できない結果が生じることがあります。ただし、define ステートメントを関数ヘッダー・ファイルに置く必要が生じる場合があります。このような場合には、他と区別できるように、定義名の前にビジネス関数名を付けてください。

例:ソース・ファイル内の#define

次の例では、ビジネス関数ソース・ファイルに define ステートメントを組み込んでいます。

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

#include <bxxxxxxx.h>

/*****************************************************************
 * Global Definitions
 *****************************************************************/
#define CACHE_GET                '1'
#define CACHE_ADD                '2'
#define CACHE_UPDATE              '3'
#define CACHE_DELETE              '4'
```

例：ヘッダー・ファイル内の#define

次の例では、ビジネス関数ヘッダーに define ステートメントを組み込んでいます。

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

#include <bxxxxxxx.h>

/************************************************************
 * Global definitions
 ************************************************************/
#define BXXXXXXX_CACHE_GET              '1'
#define BXXXXXXX_CACHE_ADD              '2'
#define BXXXXXXX_CACHE_UPDATE           '3'
#define BXXXXXXX_CACHE_DELETE           '4'
```

typedef ステートメントの使用

typedef ステートメントを使用する場合、typedef ステートメントのオブジェクトには常に大文字の記述名を指定します。データ構造体に typedef ステートメントを使用する場合は、他と区別するために必ず typedef 名にビジネス関数名を組み込んでください。次に示す、データ構造体に typedef 文を使用する例を参照してください。

例：ユーザー定義データ構造体での typedef の使用

次の例に、ユーザー定義データ構造体を示します。

```
/****************************************************************
 * 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 **/
   BOOL           bProcessed;

   MATH_NUMERIC   mnCalculatedAmount;

   JCHAR          szSummaryJob[13];

   JDEDATE        jdStartPeriodDate;

} DSX51013_INFO, *LPDSX51013_INFO;
```

関数プロトタイプの作成

関数プロトタイプを定義する際には、次のガイドラインに従ってください。

- ビジネス関数のヘッダー・ファイルにある関数プロトタイプを、必ず該当するプロトタイプ・セクションに入れます。
- ビジネス関数のソース・ファイルに関数定義を組み込み、前に関数ヘッダーを入れます。
- 関数名が本書で定義された命名規則に従っていることを確認します。
- パラメータ・リストの変数名が本書で定義された命名規則に従っていることを確認します。
- パラメータの変数名と共に関数プロトタイプのデータ・タイプを指定します。
- パラメータが 1 つのカラムに配置されるように、1 行あたり 1 つのパラメータを指定します。
- パラメータ・リストは、関数定義内で 80 文字以内とします。パラメータ・リストの分割が必要な場合は、データ・タイプと変数名の間で分割しないでください。複数行のパラメータ・リストは、最初のパラメータと位置揃えします。
- 関数ごとに戻り型を組み込みます。関数に戻り値がない場合は、戻り型としてキーワード"void"を使用します。
- 関数に何も渡されない場合は、パラメータ・リストの代わりにキーワード"void"を使用します。

参照

- ビジネス関数プロトタイプ
- 標準ヘッダー

例:ビジネス関数プロトタイプの作成

次に、標準ビジネス関数プロトタイプの例を示します。

```
/*********************************************************************
 *  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)
```

例:内部関数プロトタイプの作成

次に、標準内部関数プロトタイプの例を示します。

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


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

例:外部ビジネス関数定義の作成

次に、標準外部関数定義の例を示します。

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

例:内部関数定義の作成

次に、標準内部関数定義の例を示します。

```
/*-------------------------------------------------------------------------
 *   see sample source for standard function header
 */
void I4100040_GetSupervisorManagerDefault( LPBHVRCOM lpBhvrCom,
                                           LPSTR lpszCostCenterIn,
                                           LPSTR lpszManagerOut,
                                           LPSTR lpszSupervisorOut )
/*-------------------------------------------------------------------------
 * Note: b4100040 is the source file name
 */
{
   /*
    * internal function code
    */
}
```

変数の初期化

変数は、プログラムが使用する情報で、メモリに保管されます。テキスト文字列および数値を保管できます。

変数を宣言したら、初期化の必要があります。変数の初期化には、明示的、暗黙的の 2 つのタイプがあります。宣言ステートメントで値が割り当てられる場合、変数は明示的に初期化されます。処理中に値が変数に割り当てられる場合には、暗黙的な初期化が発生します。

ここでは、標準フォーマットの例を使用して、ビジネス関数での変数の宣言と初期化に関する標準について説明します。

変数を宣言して初期化する際には、次のガイドラインに従ってください。

- 変数の宣言には、次のフォーマットを使用します。

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

- ビジネス関数および内部関数で使用する変数は、すべてその関数の先頭で宣言します。C 言語では複合ステートメント・ブロック内で変数を宣言できますが、この標準では関数内で使用される変数を、すべて関数ブロックの先頭で宣言する必要があります。

- 同じタイプの複数の変数が存在する場合でも、変数は各行で 1 つずつ宣言します。各行をスペース 3 個分インデントし、各宣言のデータ・タイプを他のすべての変数宣言と合わせて左揃えにします。各変数名の最初の文字位置（上記フォーマット例の variable name）を、他のすべての宣言の変数名と揃えます。
- 本書で示された命名規則を使用します。変数を初期化する場合、初期値は変数のデータ・タイプに応じて任意になります。一般に、すべての変数を宣言部で明示的に初期化する必要があります。
- 説明コメントは任意です。ほとんどの場合、変数名はその変数の用途を表すように記述します。ただし、さらに説明が必要な場合や、初期値が一般的でない場合には、コメントを追加します。
- すべてのコメントは左揃えにします。
- データ構造体は、宣言セクションの直後に memset を使用して 0（ゼロ）に初期化する必要があります。
- JDE ODBC API など、一部の API には初期化ルーチンが用意されています。この場合、API と共に使用される変数は、API ルーチンを使用して初期化する必要があります。
- 必ずすべてのポインタを NULL に初期化し、該当するコール・タイプを宣言行で指定します。
- データ構造体を除き、すべての変数を宣言部で初期化します。
- 宣言したすべてのデータ構造体、MATH_NUMERIC、および JDEDATE を NULL に初期化します。
- 変数のバイト数が、保管するデータ構造体のサイズと一致することを確認します。

例：変数の初期化

次の例に、変数の初期化方法を示します。

```
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       = {0};


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

```
**************************************************************/
  HUSER            hUser          = (HUSER) NULL;
  HREQUEST         hRequestF0901  = (HREQUEST) NULL;
  DSD5100016       dsDate         = {0};
  JDEDATE          jdMidDate      = {0};


/**************************************************************
 *  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,
                  _J("4363"), (LPVOID) NULL);
     return ER_ERROR;
  }


/**************************************************************
 *  Main Processing
 **************************************************************/
  eJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                               &hUser,
                               (JCHAR *) NULL,
                               JDEDB_COMMIT_AUTO);


  memcopy ((void*)) &dsDate,jdPeriodEndDate,
          (const void*) &lpDS->jdGLDate, sizeof(JDEDATE));
```

データ構造体の初期化

テーブルへの書込み時に、テーブルでは次のデフォルト値が認識されます。

- 文字列がブランクの場合、スペース-NULL。
- 数値演算値が 0 の場合、0 値。
- 日付がブランクの場合、ブランク。
- 文字がブランクの場合、スペース。

他のビジネス関数に渡され、テーブルを更新またはフェッチするデータ構造体では、常に memset により NULL に設定します。

例：memset を使用してデータ構造体を NULL にリセット

次の例では、データ構造体を初期化時に NULL にリセットしています。

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


if ( bOpenTable )
{
   memset( (void *)(&dsF5108Key), 0x00, sizeof(KEY1_F5108) );
   jdeStrcpy( (JCHAR*) dsF5108Key.mdmcu,
              (const JCHAR*) lpDS->szBusinessUnit );
   memset( (void *)(&dsF5108), 0x00, sizeof(F5108) );


   jdeStrcpy( (JCHAR*) dsF5108.mdmcu,
              (const JCHAR*) 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)(0),
                                   (void *) (&dsF5108) );
}
```

標準変数の使用

ここでは、フラグ変数、入力/出力パラメータ、およびフェッチ変数など、標準変数の要件について説明します。

フラグ変数

フラグ変数の作成時には、次のガイドラインに従ってください。

- TRUE または FALSE として使用するフラグ変数は、ブール型（BOOL）にする必要があります。
- 質問に対して TRUE または FALSE で答えるようにフラグ変数を指定します。

次の例に、フラグ変数とその用途を示します。

bIsMemoryAllocated	メモリ割当てに適用されます。
bIsLinkListEmpty	リストをリンクします。

入力/出力パラメータ

ビジネス関数は、しばしばエラー・コードとポインタを戻します。ビジネス関数データ構造体に使用する入力/出力パラメータは、次のように指定する必要があります。

cReturnPointer	メモリを割り当てて GENLNG を戻す場合。
cErrorCode	cCallType に基づいて、失敗した場合は 1、成功した場合は 0 を戻します。
cSuppressErrorMessage	値が 1 の場合、jdeErrorSet(...)を使用してエラー・メッセージが表示されません。値が 0 の場合は、エラーが表示されます。
szErrorMessageID	エラーが発生した場合はエラー・メッセージ ID（値）を戻し、それ以外の場合はスペース 4 つを戻します。

フェッチ変数

フェッチ変数は、結果など、特定のデータを取り込んで戻したり、テーブル ID を定義したり、フェッチで使用するキーの数を指定するために使用します。

idJDEDBResult	API、または JDEDB_RESULT などの J.D. Edwards 関数。
idReturnValue	ER_WARNING または ER_ERROR などのビジネス関数の戻り値。
idTableXXXXID	XXXX は、F4101 および F41021 などのテーブル名。この変数はテーブル ID の定義に使用されます。
idIndexXXXXID	XXXX は、F4101 または F41021 などのテーブル名。この変数はテーブルのインデックス ID の定義に使用されます。
usXXXXNumColToFetch	XXXX は、F4101 および F41021 などのテーブル名。この変数はフェッチするカラムの番号です。API 関数にはパラメータとしてリテラル値を入れないでください。
usXXXXNumOfKeys	XXXX は、F4101 および F41021 などのテーブル名。この変数はフェッチに使用するキーの数です。

例:標準変数の使用

次の例に、標準変数の使用を示します。

```
/************************************************************************
 *  Variable declarations
 ************************************************************************/
 ID          idJDEDBResult   = 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      usNumColToFetch = 7;
 ushort      usNumOfKeys     = 1;


/************************************************************************
 *  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 )
   {
      idJDEDBResult = JDB_InitBhvr ((void*)lpBhvrCom,
                                    &lpdsInfo->lphUser,
                                    (JCHAR *) NULL,
                                    JDEDB_COMMIT_AUTO);
   }
   if (idJDEDBResult == JDEDB_PASSED)
   {
      idJDEDBResult = JDB_OpenTable( (*lpdsInfo->lphUser),
                                     idTableF0901,
                                     idIndexF0901,
                                     (LPID)(idFetchCol),
                                     (ushort)(usNumColFetch),
                                     (JCHAR *) NULL,
                                     &lpdsInfo->hRequestF0901 );
   }
}
/** Retrieve Account Master - AID only sent **/
if (idJDEDBResult == JDEDB_PASSED)
{
   /** Set Key and Fetch Record **/
   memset( (void *)(&dsF0901Key),
           (int) _J('¥0'), sizeof(KEY3_F0901) );
   jdeStrcpy ((char *) dsF0901Key.gmaid,
              (const JCHAR*) lpDS->szAccountID );
   idJDEDBResult = JDB_FetchKeyed ( lpdsInfo->hRequestF0901,
                                    idIndexF0901,
                                    (void *)(&dsF0901Key),
                                    (short)(1),
                                    (void *)(&dsF0901),
```

```
                                    (int)(FALSE) );
    /** Check for F0901 Record **/
    if (eJDEDBResult == JDEDB_PASSED)
    {
       statement
    }
}
```

コード化の一般ガイドライン

ここでは、適切なコードを記述するためのガイドラインについて説明します。関数呼出しの使用、メモリの割当てと解放、hRequest と hUser の使用、タイプキャスト、比較テスト、jdeStrcpy と jdeStrncpy を使用した文字列のコピー、関数エグジット・ポイントの挿入、および関数の終了について個別に説明します。

関数呼出しの使用

関数呼出しを使用して既存の関数を再利用すると、コードの重複を防ぐことができます。関数呼出しを使用する際には、次のガイドラインに従ってください。

- 各パラメータの間には必ずスペース 1 個を挿入します。
- 関数に戻り値がある場合は、関数の戻り値がエラーであるか有効値であるかを必ずチェックします。
- 他のビジネス関数の呼出しには jdeCallObject を使用します。
- 長いパラメータ・リストを指定して関数を呼び出す場合、関数呼出しは 80 文字以内にする必要があります。パラメータ・リストを 1 行または複数行に分割し、2 行目以降の最初のパラメータはパラメータ・リストの最初のパラメータと位置揃えします。
- パラメータのデータ・タイプが関数プロトタイプと一致することを確認します。データ・タイプがプロトタイプと一致しない変数を意図的に渡す場合は、そのパラメータを適切なデータ・タイプに明示的にキャストします。

外部ビジネス関数の呼出し

jdeCallObject を使用して、〈オブジェクト管理ワークベンチ〉で定義した外部ビジネス関数を呼び出します。プロトタイプ定義とデータ構造体定義を含む外部ビジネス関数用のヘッダー・ファイルをインクルードします。リターン・コードの値をチェックするようにしてください。

例:外部ビジネス関数の呼出し

次の例では、外部ビジネス関数を呼び出しています。

```
/*------------------------------------------------------------
 *
 *  Retrieve account master information
 *
 *------------------------------------------------------------*/
   idReturnCode = jdeCallObject(_J("ValidateAccountNumber"),
                                NULL,
                                lpBhvrCom,
                                lpVoid,
```

```
                        (void*) &dsValidateAccount,
                        (CALLMAP*) NULL,
                        (int) 0,
                        (JCHAR*) NULL,
                        (JCHAR*) NULL,
                        (int) 0 );

  if ( idReturnCode == ER_SUCCESS )
  {
     statement;
  }
```

内部ビジネス関数の呼出し

内部関数は、他のビジネス関数で呼び出さないでください。コードをソース間でコピーする場合は、すべての内部関数名を、関数に関する現行の内部命名標準に従って変更します。他のビジネス関数の内部関数を別のソース・コードからインクルードすることはできますが、クライアント/サーバーの場合、現行のソース・コードの外側では使用しないでください。

ビジネス関数間でのポインタの受渡し

ビジネス関数間では、ポインタを直接渡さないでください。ポインタのメモリ・アドレスは 32 ビット以内とします。プラットフォーム上で、32 ビットしかサポートしていないクライアントに、32 ビットを超えるポインタのアドレスを渡すと、有効桁数が切り捨てられてアドレスが無効になる可能性があります。

ビジネス関数間でポインタを共有するには、アドレスを配列に格納する必要があります。この配列は、オブジェクト構成マネージャ(OCM)で指定したサーバー・プラットフォーム上に置かれます。配列では、メモリ位置を最大 100 まで割り当てて保管できます。配列の管理には、J.D. Edwards ソフトウェア・ツールを使用します。配列内の位置のインデックスは long integer 型または ID です。このインデックスをビジネス関数間で渡すには、ビジネス関数データ構造体で GENLNG データ辞書オブジェクトを使用します。

配列へのアドレスの保管

後で取り込むために、jdeStoreDataPtr を使用して、割り当てたメモリ・ポインタを配列に保管します。配列内の位置のインデックスが戻されます。インデックスは、ビジネス関数データ構造体(lpDS)を介して渡す必要があります。

例:配列へのアドレスの保管

次の例に、アドレスを配列に保管する方法を示します。

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

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



}
```

配列からのアドレスの取込み

jdeRetrieveDataPtr を使用して、現在のビジネス関数外でアドレスを取り込みます。配列内のポインタのインデックスは、ビジネス関数データ構造体(lpDS)を介して渡す必要があります。
jdeRetrieveDataPtr を使用すると、アドレスは配列内に残り、後で再び取り込むことができます。

例:配列からのアドレスの取込み

次の例では、配列からアドレスを取り込んでいます。

```
  lpdsF43199 = (LPF43199) jdeRetrieveDataPtr
                (hUser, lpDS->idF43199Pointer);
```

配列からのアドレスの削除

jdeRemoveDataPtr を使用して、配列セルからアドレスを削除し、その配列セルを解放します。配列内のポインタのインデックスは、ビジネス関数データ構造体(lpDS)を介して渡す必要があります。
jdeStoreDataPtr ごとに、対応する jdeRemoveDataPtr 呼出しが存在する必要があります。メモリの割当てに jdeAlloc を使用する場合、メモリの解放には jdeFree を使用します。

例:配列からのアドレスの削除

次の例では、配列からアドレスを削除しています。

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

```
  if (lpGenericPtr != (void *) NULL)
  {
     jdeFree((void *)lpGenericPtr);
     lpDS->idGenericLong = (ID) 0;
     lpGenericPtr = (void *) NULL;
  }

}
```

メモリの割当てと解放

jdeAlloc を使用してメモリを割り当てます。jdeAlloc はパフォーマンスに影響するので、多用しないでください。

jdeFree を使用して、ビジネス関数内のメモリを解放します。jdeAlloc ごとに、メモリを解放するための jdeFree が存在する必要があります。

注:

イベント・ルール・ロジックを介してメモリを解放するには、ビジネス関数 Free Ptr To Data Structure、B4000640 を使用します。

例:ビジネス関数内のメモリの割当てと解放

次の例では、jdeAlloc を使用してメモリを割り当ててから、関数クリーンアップ・セクションで jdeFree を使用して、そのメモリを解放しています。

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

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

hRequest および hUser の使用

一部の API 呼出しには、hUser および hRequest が必要です。hUser を取得するには JDBInitBhvr を使用し、hRequest を取得するには JDBOpenTable を使用します。hUser および hRequest を、変数宣言行で NULL に初期化します。すべての宣言について、関数クリーンアップ・セクションで JDB_CloseTable( )および JDB_FreeBhvr( )を設定する必要があります。

タイプキャスト

タイプキャストは、型変換と呼ぶこともあります。タイプキャストを使用するのは、関数が特定のタイプの値を必要とする場合、関数パラメータを定義する場合、および jdeAlloc()を使用してメモリを割り当てる場合です。

注:

この標準は、すべての関数および関数プロトタイプに適用されます。

比較テスト

比較には、常に明示テストを使用します。比較テストには代入を埋め込まないでください。変数に値または結果を割り当て、その変数を比較テストに使用します。

浮動小数点変数は、常に<=または>=を使用してテストします。一部の浮動小数点数値を正確に表すことができないため、==や!=は使用しないでください。

例:比較テスト

次の例は、比較テスト用の C 言語コードを作成する方法を示します。

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


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

例:ブール論理を使用する TRUE/FALSE 比較テストの作成

次の例に、ブール論理を使用する TRUE/FALSE 比較テストを示します。

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

jdeStrcpy と jdeStrncpy を使用した文字列のコピー

ビジネスユニットなど、同じ長さの文字列をコピーする場合は、jdeStrcpy ANSI API を使用できます。記述など、文字列の長さが異なる場合は、jdeStrncpy ANSI API を使用して、後続 NULL 文字に相当する 1 を差し引いて文字数を指定します。

```
/*************************************************************
 * Variable Definitions
 *************************************************************/
   JCHAR          szToBusinessUnit(13);
   JCHAR          szFromBusinessUnit(13);
   JCHAR          szToDescription(31);
   JCHAR          szFromDescription(41);

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

   jdeStrcpy((JCHAR *) szToBusinessUnit,
             (const JCHAR *) szFromBusinessUnit );

   jdeStrncpy((JCHAR *) szToDescription,
              (const JCHAR *) szFromDescription,
              DIM(szToDescription)-1 );
```

関数クリーンアップ領域の使用

関数クリーンアップ領域を使用して、hRequest および hUser など、割当済みのメモリを解放します。

例:関数クリーンアップ領域を使用したメモリの解放

次の例に、関数クリーンアップ領域でメモリを解放する方法を示します。

```
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 ) ;
```

関数エグジット・ポイントの挿入

可能な場合には、関数からの単一のエグジット・ポイント(戻り)を使用します。ビジネス関数に単一のエグジット・ポイントがある方が、コードが構造化されます。また、単一のエグジット・ポイントを使用すると、プログラマは戻り直前にメモリの解放や ODBC 要求の終了などのクリーンアップを実行できます。より複雑な関数では、この操作が難しい場合や、不可能な場合があります。関数の途中にエグジット・ポイントをプログラミングする場合は、メモリの解放や ODBC 要求の終了など、必要なクリーンアップ・ロジックを組み込みます。

関数の戻り値を使用して、ステートメントの実行を制御します。ビジネス関数の戻り値として、ER_SUCCESS または ER_ERROR のどちらか一方を使用できます。関数の戻り値を ER_SUCCESS に初期化すると、それを使用して処理フローを判別できます。

例：関数へのエグジット・ポイントの挿入

次の例に、関数の戻り値を使用してステートメントの実行を制御する方法を示します。

```
  ID                idReturn              = ER_SUCCESS;


/*****************************************************************
 * Main Processing
 *****************************************************************/
  memset( (void *)(&dsInfo), 0x00, 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 == _J('Y')) )
  {
    /** Process All Periods 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)
      {
        statement;
      }
    } /* End Processing */
```

```
  }

/*********************************************************************
 * Function Clean Up
 *********************************************************************/
  if ( (dsInfo.hUser) != (HUSER) NULL )
  {
     statement;
  }


  return idReturn;
```

関数の終了

関数の終了時には、常に値を戻します。

移植性

移植性とは、プログラムを無修正のまま複数のコンピュータ上で動作させる能力です。J.D. Edwards は移植性を備えた環境です。ここでは、システム間でオブジェクトを移植する場合の考慮事項とガイドラインについて説明します。

リレーショナル・データベース・システムの開発に影響を及ぼす標準は、次の標準によって決まります。

- ANSI (American National Standards Institute)標準
- X/OPEN (European body)標準
- 国際標準化機構(ISO: International Organization for Standardization)の SQL 標準

業界標準により、異なるリレーショナル・データベース・システムを同じように処理できるのが理想的です。主要なベンダー各社は業界標準をサポートしていますが、SQL 言語の機能を強化する拡張機能も提供しています。さらにベンダー各社は、自社製品のアップグレードと新バージョンを常にリリースしています。

このような機能拡張とアップグレードは移植性に影響します。ソフトウェア開発が産業に及ぼす影響により、アプリケーションには、データベース・ベンダー間の違いによって左右されない、データベースへの標準インターフェイスが必要です。ベンダーが新しいリリースを提供する場合は、既存のアプリケーションに及ぼす影響を最小限に抑える必要があります。移植性に関する問題を解決するために、多くの組織がオープン・データベース・コネクティビティ(ODBC)と呼ばれる標準データベース・インターフェイスに移行しています。

移植性のガイドライン

移植性標準に準拠するビジネス関数を開発する際には、次のガイドラインに従ってください。

- ビジネス関数は、移植性を確保するために ANSI 互換にする必要があります。コンピュータ・プラットフォームごとに制限事項が異なるため、このルールには例外が存在します。ただし、非 ANSI 準拠のビジネス関数を使用する場合は、必ずビジネス関数標準委員会の承認を受けてください。
- データ配置に依存するプログラムは、移植性がないため作成しないでください。これは、データの配置方法が、バイトの割当てやワードの割当てなど、システムごとに異なるためです。たとえば、1 バイトである 1 文字のフィールドがあるとします。このフィールドに 1 バイトしか割り当てないシステムや、ワード全体を割り当てるシステムがあります。
- ベンダーのライブラリと関数呼出しはシステムに依存しており、そのベンダー専用であることに注意してください。つまり、プログラムのコンパイルに異なるコンパイラを使用すると、その関数がエラーになります。
- ポインタ演算はシステムに依存しており、データ配置に基づいているため、使用時には注意してください。
- どのシステムでも変数の初期化方法が同じであるとは想定しないでください。変数は常に明示的に初期化してください。
- データ構造体内でオフセットを使用してオブジェクトを取り込むときには、注意してください。このガイドラインはデータ配置にも関連しています。オフセットはキャッシュ・インデックスの定義に使用してください。

- パラメータが関数パラメータと一致しない場合は、常にタイプキャストを行います。

注:
関数宣言では、JCHAR szArray[13]は(JCHAR *)とは異なります。したがって、特定の関数で使用する場合は、szArray で(JCHAR *)のタイプキャストが必要です。

- データが失われることがあるため、代入ステートメントの左辺ではタイプキャストしないでください。たとえば、ステートメント(short)nValue = (long)の場合、long integer 型の値が大きすぎて short integer データ型に収まらないと、lValue では値が失われます。
- C++のコメントを使用しないでください(C++のコメントは 2 つのスラッシュで始まります)。

一般的なサーバー・ビルド・エラーと警告

ERP ビジネス関数は、移植性を確保するために ANSI 互換にする必要があります。コンピュータ・プラットフォームやサーバーごとに独自の制限事項があるため、ビジネス関数はすべてのサーバー標準に準拠する必要があります。ここでは、各種サーバー上で正常にビルドされるビジネス関数をコード化するためのガイドラインについて説明します。

コメント内のコメント

コメントに他のコメントを挿入しないでください。各"/*"には、対応する"*/"が必要です。次の例を参照してください。

例:ANSI 標準に準拠する C 言語コメント

次の C 言語標準コメント・ブロックを使用します。

```
/*************************************************************************
 * Correct Method of C Comments                                          *
 *************************************************************************/
/* SAR 1234567 Begin*/
/* Populate the lpDS->OrderedPlacedBy value from the userID only in
   the ADD mode */
   if ( lpDS->cHeaderActionCode == _J('1'))
   {
      if (IsStringBlank(lpDS->szOrderedPlacedBy))
       {
           jdeStrcpy((JCHAR *)(lpDS->szOrderedPlacedBy),
                    (const JCHAR *)(lpDS->szUserID));
      }/* End of defaulting in the user id into Order placed by
           if the later was left blank */
```

```
  }/* SAR 1234567 End */
```

例：異なるサーバー上でエラーとなるコメント内のコメント

次の例に、異なるサーバー上でエラーとなる可能性のあるコメント内のコメントを示します。

```
/*****************************************************************************
 C Comments within Comments Causing Server Build Errors and Warnings
 *****************************************************************************/
/* SAR 1234567 Begin
/* Populate the lpDS->OrderedPlacedBy value from the userID only in
  the ADD mode */
*/
  if ( lpDS->cHeaderActionCode == _J('1'))
  {
    if (IsStringBlank(lpDS->szOrderedPlacedBy))
      {
        jdeStrcpy((JCHAR *)(lpDS->szOrderedPlacedBy),
                (const JCHAR *)(lpDS->szUserID));
  }/* End of defaulting in the user id into the Order placed by
   /* if the later was left blank */
  }/* SAR 1234567 End */
```

ビジネス関数末尾の改行文字

一部のサーバーでは、正常にビルドするためにソース・ファイルとヘッダー・ファイルの終わりに改行文字を必要とします。各ビジネス関数の末尾に改行文字を 1 個追加するのがベスト・プラクティスです。コードの末尾で[Enter]キーを押して改行文字を追加します。

NULL 文字の使用

NULL 文字 '¥0' は慎重に使用してください。この文字はバックスラッシュで始まります。'/0' を使用するとエラーになりますが、コンパイラではレポートされません。

例：NULL 文字の使用

次の例に、NULL 文字の誤った使用と正しい使用を示します。

```
/*************Initialize Data Structures*************************/
/*Error Code*/
/* ‘/0’ is used assuming it to be a NULL character*/
/* memset((void *)(&dsVerifyActivityRulesStatusCodeParms),
        (int)('/0'), sizeof(DSD4000260A));*/



/*Correct Use of NULL Character*/
memset((void *)(&dsVerifyActivityRulesStatusCodeParms),
     (int)('¥0'), sizeof(DSD4000260A));
```

include ステートメントでの小文字

外部ビジネス関数またはテーブルをヘッダー・ファイルにインクルードする場合は、include ステートメントに小文字を使用します。大文字を使用するとビルド・エラーになります。

例：include ステートメントでの小文字の使用

次の例に、include ステートメントでの小文字の間違った使用と正しい使用を示します。

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



/*Incorrect method of including external business function header*/
/*Include Statement Causing Build Warnings on Various Servers*/
  #include <B0000130.h>



/*Correct method of including external business function header*/


  #include <b0000130.h>
```

参照されない初期化済み変数

ビジネス関数の変数宣言セクションで宣言され、初期化される変数は、それぞれプログラム内で使用する必要があります。たとえば、変数 idReturnValue が初期化されている場合は、それをプログラムのどこかで使用する必要があります。

J.D. Edwards 定義の構造

J.D. Edwards ソフトウェアには、ビジネス関数の作成時に注意すべき 2 つのデータ・タイプ、MATH_NUMERIC と JDEDATE が用意されています。この 2 つのデータ・タイプは変更の可能性があるため、J.D. Edwards ソフトウェアに用意されている共通ライブラリ API を使用して操作してください。この 2 つのデータ・タイプのメンバには直接アクセスしないでください。

MATH_NUMERIC データ・タイプ

MATH_NUMERIC データ・タイプは、一般に J.D. Edwards ソフトウェアで数値を表すために使用されます。このデータ・タイプは次のように定義されます。

```
struct tag MATH_NUMERIC


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


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

MATH_NUMERIC 要素	説明
String	区切り記号なしの数字。
Sign	マイナス符号は数値が負であることを示します。それ以外の場合、値は 0x00 になります。
EditCode	数値の表示形式を設定するデータ辞書の編集コード。
nDecimalPosition	小数点以下の桁数。
nLength	String の桁数。

wFlags	処理フラグ。
szCurrency	通貨コード。
nCurrencyDecimals	通貨の小数部の桁数。
nPrecision	データ辞書サイズ。

MATH_NUMERIC 変数を割り当てる場合は、MathCopy API を使用します。MathCopy により、Currency などの情報がポインタの位置にコピーされます。この API を使用すれば、代入時にデータが失われることはありません。

ローカルの MATH_NUMRIC 変数は ZeroMathNumeric API を使用して初期化します。
MATH_NUMERIC が初期化されていないと、データ構造に無効な情報、特に無効な通貨情報が含まれる可能性があり、実行時に予期しない結果となることがあります。

```
/*************************************************
 * Variable Definitions
 *************************************************/
   MATH_NUMERIC      mnVariable   = {0};


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

JDEDATE データ・タイプ

JDEDATE データ・タイプは、一般に J.D. Edwards ソフトウェアで日付を表すために使用されます。このデータ・タイプは次のように定義されます。

```
struct tag JDEDATE
{
   short nYear;
   short nMonth;
   short nDay;
};


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

JDEDATE 要素	説明
nYear	年
nMonth	月
nDay	日

参照

- エラー・メッセージとワークフロー・メッセージの作成については、『開発ツール』ガイドの「メッセージ処理」

memcpy を使用した JDEDATE 変数の割当て

JDEDATE 変数を割り当てる場合は、memcpy 関数を使用します。memcpy 関数は、情報をポインタの位置にコピーします。ポインタを使用しない割当てを行うこともできますが、そこで割り当てたローカル変数のスコープが失われ、結果としてデータが失われることがあります。

```
/*******************************************************
 * Variable Definitions
 *******************************************************/
  JDEDATE      jdToDate;


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

JDEDATECopy

JDEDATECopy を使用すると、memcpy の場合と同様に JDEDATE 変数を割り当てることができます。構文は次のとおりです。

```
#define JDEDATECopy(pDest, pSource)
            memcpy(pDest, pSource, sizeof(JDEDATE))
```

エラー・メッセージ

メッセージは、情報をエンドユーザーに伝える効果的で使用しやすい手段です。簡単なメッセージまたはテキスト置換メッセージを使用できます。

テキスト置換メッセージは、ユーザーに特定の情報を提示します。実行時には、メッセージに含まれる変数が置換値に置き換えられます。テキスト置換メッセージには、次の 2 種類があります。

- エラー・メッセージ（用語解説グループ E）
- ワークフロー・メッセージ（用語解説グループ Y）

JDB および JDE キャッシュ API からの戻りコードをすべて確認し、呼び出している関数に該当するエラー・メッセージを設定するか、戻すか、またはその両方を行う必要があります。次の表に、JDB および JDE キャッシュ・エラーに使用する標準的なエラー・メッセージを示します。

JDB エラーは次のとおりです。

エラーID	説　明
078D	テーブルのオープンに失敗しました。
078E	テーブルのクローズに失敗しました。
078F	テーブルへの挿入に失敗しました。
078G	テーブルからの削除に失敗しました。
078H	テーブルの更新に失敗しました。
078I	テーブルからの取込みに失敗しました。
078J	テーブルからの選択に失敗しました。
078K	テーブルの順序設定に失敗しました。
078S *	動作の初期化に失敗しました。

* 078S はテキスト置換を使用しません。

JDE キャッシュ・エラーは次のとおりです。

エラーID	説　明
078L	キャッシュの初期化に失敗しました。
078M	カーソルのオープンに失敗しました。
078N	キャッシュからの取込みに失敗しました。
078O	キャッシュへの追加に失敗しました。

078P　　キャッシュの更新に失敗しました。

078Q　　キャッシュからの削除に失敗しました。

078R　　キャッシュの終了に失敗しました。

lpDS へのエラー・メッセージ用パラメータの挿入

エラー・メッセージ処理では、lpDS にパラメータ cSuppressErrorMessage および szErrorMessageID を挿入する必要があります。ここでは、それぞれの機能について説明します。

- cSuppressErrorMessage (SUPPS)

  有効なデータは 1 または 0 です。ビジネス関数に jdeErrorSet(...)が使用される場合は、このパラメータが必須です。cSuppressErrorMessage を 1 に設定すると、エラーは設定されません。これは、jdeErrorSet によりエラー・メッセージが自動的に表示されるためです。

- szErrorMessageID (DTAI)

  この文字列には、ビジネス関数から戻されるエラー・メッセージ ID 値が含まれます。ビジネス関数でエラーが発生する場合、szErrorMessageID にはそのエラー番号 ID が含まれます。

注:

szErrorMessageID は、関数の先頭でスペース 4 個に初期化する必要があります。初期化しないとメモリ・エラーになることがあります。

例:lpDS のエラー・メッセージ用パラメータ

次の例には、lpDS パラメータである cSuppressErrorMessage および szErrorMessageID が含まれています。

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


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

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


   return idReturnValue;
```

標準ビジネス関数のエラー状態

J.D. Edwards データベース API を使用するビジネス関数では、データベース関数を呼び出す前に Initialize Behavior 関数を呼び出す必要があります。Initialize Behavior 関数が正常終了しない場合は、エラー078S が設定されます。

例:動作初期化エラー

次の例に、動作初期化エラーを示します。

```
/************************************************************************
 * Initialize Behavior
 ************************************************************************/
idJDBReturn = JDB_InitBhvr(lpBhvrCom,
                           &hUser,
                           (JCHAR *) NULL,
                           JDEDB_COMMIT_AUTO);


if (idJDBReturn != JDEDB_PASSED)
{
   jdeStrcpy (lpDS->szErrorMessageID, _J("078S"));
   if (lpDS->cSuppressErrorMessage != _J('1'))
   {
      jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _J("078S"), (LPVOID) NULL);
   }
   return ER_ERROR;
}
```

テキスト置換を使用した特定のエラー・メッセージの表示

ビジネス関数でエラー・メッセージを戻すために、J.D. Edwards のテキスト置換 API を使用できます。テキスト置換は、特定のエラー・メッセージを表示する柔軟性のある方法です。

テキスト置換は、データ辞書を介して行われます。テキスト置換を使用するには、まず特定のエラー・メッセージ用のテキスト置換を定義するデータ辞書項目を設定する必要があります。JDB および JDE キャッシュのエラー・メッセージについては、既に選択項目が作成されており、この章のリストに記載してあります。

キャッシュおよびテーブルに関するエラー・メッセージは、CNC（コンフィギュラブル・ネットワーク）アーキテクチャでは重要です。C のプログラマは、テーブルまたはキャッシュ API を処理する際に、適切なエラー・メッセージを設定する必要があります。

JDB API エラーでは、API が失敗したテーブルの名称を置換します。JDE キャッシュ API エラーでは、API が失敗したキャッシュの名称を置換します。

テキスト置換を使用するエラーを呼び出す場合は、次の方法を使用する必要があります。

- エラー・メッセージ内で置換する情報を含んだデータ構造体をロードします。
- jdeErrorSet を呼び出してエラーを設定します。

例：エラー・メッセージでのテキスト置換

次の例では、JDB_OpenTable にテキスト置換を使用しています。

```
/*****************************************************************
 * Open the General Ledger Table F0911
 *****************************************************************/
eJDBReturn = JDB_OpenTable( hUser,
                            ID_F0911,
                            ID_F0911_DOC_TYPE_NUMBER__B,
                            idColF0911,
                            nNumColsF0911,
                            (JCHAR *)NULL,
                            &hRequestF0911);

if (eJDBReturn != JDEDB_PASSED)
{
   memset((void *)(&dsDE0022), 0x00, sizeof(dsDE0022));
   jdeStrncpy((JCHAR *)dsDE0022.szDescription,
              (const JCHAR *)(_J("F0911")),
              DIM(dsDE0022.szDescription)-1);
```

```
  jdeErrorSet (lpBhvrCom, lpVoid,(ID)0, _J("078D"), &dsDE0022);
}
```

DSDE0022 データ構造体

データ構造体 DSDE0022 には、単一の項目 szDescription[41]が含まれています。jdeErrorSet の最後のパラメータとして、JDB エラーと JDE キャッシュ・エラーに DSDE0022 データ構造体を使用します。

データ構造体エラーと jdeCallObject とのマッピング

他のビジネス関数を呼び出すビジネス関数には、常に jdeCallObject を使用する必要があります。jdeCallObject を使用する場合は、エラーID を一致させてください。

プログラマは、元のビジネス関数からの ID を、jdeCallObject 内のビジネス関数のエラーID と一致させる必要があります。データ構造体を jdeCallObject 内で使用して、このタスクを完了します。

```
/************************************************************
 * Variable declarations
 ************************************************************/
CALLMAP      cm_D0000026[2]   = {{IDERRmnDisplayExchgRate_62,
                          IDERRmnExchangeRate_2}};
ID           idReturnCode     = ER_SUCCESS;   /* Return Code */
/************************************************************
 * Business Function structures
 ************************************************************/
DSD0000026   dsD0000026       = {0};       /* Edit Tolerance */


  idReturnCode = jdeCallObject(_J("EditExchanbeRateTolerance"),
                           NULL,
                           lpBhvrCom,
                           lpVoid,
                           (void *)&dsD0000026,
                           (CALLMAP *)&cm_D0000026,
                           ND0000026,
                           (JCHAR *)NULL,
                           (JCHAR *)NULL,
                           (int)0);
```

データ辞書トリガー

データ辞書トリガーは、データ辞書項目に編集/表示ロジックを関連付ける際に使用します。フォーム内の項目にアクセスすると、アプリケーションのランタイム・エンジンにより、データ辞書に関連付けられているトリガーが実行されます。

カスタムのデータ辞書トリガーは C または NER で記述し、正常に実行するには特定のデータ構造体が必要です。カスタム・トリガーのデータ構造体は、3 つの事前定義済みのメンバと 1 つの変数メンバで構成されます。事前定義済みのメンバは、すべてのカスタム・トリガーに共通です。変数メンバはトリガーごとに異なり、データ辞書項目に関連付けられた特定のデータ要素を使用して作成します。

次の表に、データ構造体メンバの順序および各メンバのエイリアスと説明を示します。

構造体メンバ名	エイリアス	説　明
idBhvrErrorId	BHVRERRID	トリガー関数でアプリケーションにエラー状況(ER_ERROR or ER_SUCCESS)を戻すために使用します。
szBehaviorEditString	BHVREDTST	アプリケーションのランタイム・エンジンで、データ辞書フィールドの値をトリガー関数に渡すために使用します。
szDescription001	DL01	トリガー関数で、値の記述をアプリケーションに戻すために使用します。
szHomeCompany, mnAddressNumber	HMCO、AN8	トリガー関数でエラー（CALLMAP フィールド）を設定するために使用します。

例：カスタムのデータ辞書トリガー

編集トリガーIsColumnInAddressBook は、カスタムのデータ辞書トリガーの一例です。このトリガーは、szBehaviorEditString で渡された住所番号の住所録レコードが存在するかどうかを検証し、存在する場合は szDescription001 で名称を戻します。このトリガーの変数メンバは mnAddressNumber で、AN8 データ辞書項目を使用して作成されています。

このトリガーの C プログラムでは、次の構造体を使用します。

```
typedef struct tagDSD0100039
{
   ID                    idBhvrErrorId;
   JCHAR                 szBehaviorEditString[51];
   JCHAR                 szDescription001[31];
   MATH_NUMERIC          mnAddressNumber;
} DSD0100039,FAR *LPDSD0100039;
```

Unicode 準拠標準

Unicode 標準は、文字とテキストに関する汎用文字コード体系です。この標準では、テキスト・データを各国間でやりとりし、グローバル・ソフトウェアの基盤を形成できるように、複数言語によるテキストをエンコードするための一貫した方法が定義されています。

ここでは Unicode について説明します。

- Unicode は、事実上あらゆる言語の文字を含む超大型キャラクタ・セットです。
- Unicode は 1 文字に 2 バイトを使用します。

  2 バイトを使用して最大 64,000 文字までサポートできます。また、Unicode にはサロゲートと呼ばれるメカニズムも用意されており、2 バイトのペアを使用してさらに 100 万文字を記述します。

- 0x00 は 1 文字における有効バイトです。

  たとえば、文字 A は 0x00 0x41 として記述されます。つまり、strlen()や strcpy など、通常の文字列関数は、Unicode データを処理しません。

したがって、データ・タイプ char は使用せず、Unicode 文字には JCHAR、非 Unicode 文字には ZCHAR を使用してください。また、非 Unicode API とのインターフェイスを必要とするコードには、char の代わりに ZCHAR を使用します。

旧構文 廃止	**新規構文 非 Unicode**	**新規構文 Unicode**
Char	ZCHAR	JCHAR
char *、PSTR	ZCHAR*、PZSTR	JCHAR*、PJSTR
'A'	_Z('A')	_J('A')
"string"	_Z("string")	_J("string")

Unicode 文字列関数

すべての文字列関数には、Unicode バージョンと非 Unicode バージョンの 2 つが存在します。Unicode と非 Unicode の文字列関数の命名規則は、次のとおりです。

jdeSxxxxxx()は、Unicode 文字列関数を示します。

jdeZSxxxx()は、非 Unicode 文字列関数を示します。

次のような置換関数があります。

旧文字列関数	新文字列関数 非 Unicode	新文字列関数 Unicode
strcpy()	jdeZStrcpy()	jdeStrcpy()
strlen()	jdeZStrlen()	jdeStrlen()
strstr()	jdeZStrstr()	jdeStrstr()
sprintf()	jdeZSprintf()	jdeSprintf()
strncpy()	jdeZStrncpy()	jdeStrncpy()

注:

jdestrcpy()関数は、Unicode への移行前に使用されていました。Unicode でも、従来の jdestrcpy()が jdeStrncpyTerminate()に変更されています。今後、開発者はこれまで jdestrcpy()を使用していた箇所に jdeStrncpyTerminate()を使用する必要があります。

strcpy、strlen、printf など、従来の文字列関数は使用しないでください。文字列はすべての jdeStrxxxxxx 関数で明示的に処理されるため、バイト数を戻す sizeof()演算子の代わりに文字数を使用します。

jdeStrncpy()を使用する場合、3 番目のパラメータはバイト数ではなく文字数となります。

DIM()マクロは、配列の文字数を示します。たとえば、"JCHAR a[10];", DIM(a)は 10 を戻しますが、"JCHAR a[10];", DIM(a)は 20 を戻します。したがって、"strncpy (a, b, sizeof (a));"は jdeStrncpy (a, b, DIM (a));"とする必要があります。

参照

- Unicode 文字列関数の全リストについては、『Unicode Codebook』
-

例:Unicode 文字列関数の使用

次の例に、Unicode 文字列関数の使用方法を示します。

```
/*************************************************************************
  In this example jdeStrncpy replaces strncpy. Also sizeof is
  replaced by DIM.
 *************************************************************************/
/* Set key to F38112 */


/*Unicode Compliant*/
jdeStrncpy(dsKey1F38112.dxdcto,
```

```
        (const JCHAR *)(dsF4311ZDetail->pwdcto),
          DIM(dsKey1F38112.dxdcto) - 1);
```

Unicode メモリ関数

memset()関数は、メモリを 1 バイトずつ変更します。たとえば、memset (buf, ' ', sizeof (buf));は、最初の引数 buf が指す 10 バイトを値 0x20、つまりスペースに設定します。Unicode 文字は 2 バイトのため、各文字は 0x2020、つまり Unicode におけるデバッガ文字(†)に設定されます。

新しい関数 jdeMemset()は、メモリ文字をバイト単位ではなく文字単位で設定します。この関数は void ポインタ、JCHAR および設定バイト数を取ります。jdeMemset (buf, _J (' '), sizeof (buf));を使用して、各文字が 0x0020 になるように Unicode 文字列 buf を設定してください。jdeMemset()を使用する場合、3 番目のパラメータ sizeof(buf)は文字数ではなくバイト数です。

注:

メモリ・ブロックに NULL を埋め込む場合には memset を使用できます。他のすべての文字については jdeMemset を使用してください。また、jdeMemset は NULL 文字にも使用できます。

例:文字を NULL 以外の値に設定する際の jdeMemset の使用

次の例に、文字を NULL 以外の値に設定する際の jdeMemset の使用方法を示します。

```
/*****************************************************************************
  In this example memset is replaced by jdeMemset. We need to change
  memset to jdeMemset because we are setting each character of the
  string to a value other than NULL.  Also, because jdeMemset works in
  bytes we cannot just subtract 1 from sizeof(szSubsidiaryBlank) to
  prevent the last character from being set to ' '.  We must multiply
  1 by sizeof(JCHAR).
 *****************************************************************************/


/*Unicode Compliant*/
jdeMemset((void *)(szSubsidiaryBlank), _J(' '),
          (sizeof(szSubsidiaryBlank) - (1*sizeof(JCHAR))));
```

ポインタ演算

JCHAR ポインタを送る場合は、適切な数だけ送る必要があります。次の例は、JCHAR 文字列を構成する配列の各メンバをブランクに初期化することを意図しています。For ループ内では、ポインタは JCHAR 文字列の配列メンバの 1 つに値を代入した後、次のメンバを指すように送られます。そのためには、ポインタに文字列の最大長を追加します。pStringPtr は JCHAR へのポインタとして定義されているため、pStringPtr に MAXSTRLENGTH を追加すると、pStringPtr は文字列配列の次のメンバを指すことになります。

```
#define  MAXSTRLENGTH 10
JCHAR              *pStringPtr;
LPMATH_NUMERIC     pmnPointerToF3007;
for(i=(iDayOfTheWeek+iNumberOfDaysInMonth);i<CALENDARDAYS;i++)
 {
    FormatMathNumeric(pStringPtr, &pmnPointerToF3007[i]);
    pStringPtr = pStringPtr + MAXSTRLENGTH;
 }
```

次の表に、pStringPtr に MAXSTRLENGTH を追加した場合の効果を示します。どちらの表の場合も、最初の行はメモリ位置で、最後の行はそのメモリ位置の内容です。

矢印は、MAXSTRLENGTH を追加する前の pStringPtr が指すメモリ位置を示します。

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	49	00	52	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20

21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
00	49	00	52	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20

次の矢印は、MAXSTRLENGTH を追加した後の pStringPtr が指すメモリ位置を示します。
pStringPtr に 10 を追加すると、タイプ JCHAR が宣言されているため 20 バイト移動します。

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	52	00	53	00	54	00	20	00	20	00	20	00	20	00	20	00	20	00	20

.

21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
00	49	00	52	00	20	00	20	00	20	00	20	00	20	00	20	00	20	00	20

If pStringPtr が値 MAXSTRLENGTH * sizeof(JCHAR)だけ送られると、pStringPtr は意図したより 2 倍移動するためメモリが破損します。

オフセット

ポインタにオフセットを追加して他の変数やエンティティの位置を導出する際には、最初にオフセットが作成された方法を判断することが重要になります。

次の例では、Cache Key セグメントの位置に達するように、lpData に lpKeyStruct->CacheKey[n].nOffset が追加されています。このオフセットは、バイト数を戻す ANSI C 関数 offsetof を使用して作成されたセグメントに関するものでした。したがって、Cache Key セグメントの位置に達するには、データ構造体ポインタをタイプ BYTE にキャストします。

```
lpTemp1 = (BYTE *)lpData + lpKeyStruct->CacheKey[n].nOffset;

lpTemp2 = (BYTE *)lpKey + lpKeyStruct->CacheKey[n].nOffset;
```

非 Unicode 環境では、文字サイズが 1 バイトであるため、lpData はタイプ CHAR *にキャストされている場合があります。ただし、Unicode 環境では、JCHAR のサイズが 2 バイトであるため、lpData はタイプ(JCHAR *)に明示的にキャストする必要があります。

MATH_NUMERIC API

MATH_NUMERIC データ構造体の文字列メンバは、ZCHAR(非 Unicode)フォーマットになっています。J.D. Edwards 共通ライブラリ API には、これらの文字列を JCHAR(Unicode)および ZCHAR(非 Unicode)フォーマットで取り込んで操作する関数が複数用意されています。

MATH_NUMERIC データ・タイプの文字列値を JCHAR フォーマットで取り込むには、FormatMathNumeric API 関数を使用します。次に、この関数の使用例を示します。

```
/* Declare variables */

JCHAR                   szJobNumber[MAXLEN_MATH_NUMERIC+1] = _J("¥0");

/* Retrieve the string value of the job number */
```

```
FormatMathNumeric(szJobNumber, &lpDS->mnJobnumber);
```

MATH_NUMERIC データ・タイプの文字列値を ZCHAR フォーマットで取り込むには、jdeMathGetRawString API 関数を使用します。次に、この関数の使用例を示します。

```
/* Declare variables */

ZCHAR                   zzJobNumber[MAXLEN_MATH_NUMERIC+1] = _Z("¥0");

/* Retrieve the string value of the job number */

zzJobNumber = jdeMathGetRawString(&lpDS->mnJobnumber);
```

一般に使用されるもう 1 つの MATH_NUMERIC API 関数が、jdeMathSetCurrencyCode です。この関数は、MATH_NUMERIC データ構造体の通貨コード・メンバを更新するために使用します。この関数には、jdeMathCurrencyCode および jdeMathCurrencyCodeUNI という 2 つのバージョンがあります。jdeMathCurrencyCode 関数は通貨コードを ZCHAR 値で更新する場合に使用し、jdeMathCurrencyCodeUNI 関数は通貨コードを JCHAR 値で更新する場合に使用します。次に、この 2 つの関数の使用例を示します。

```
/* Declare variables */

ZCHAR                   zzCurrencyCode[4] = _Z("USD");

JCHAR                   szCurrencyCode[4] = _J("USD");

/* Set the currency code using a ZCHAR value */

jdeMathSetCurrencyCode(&lpDs->mnAmount, (ZCHAR *) zzCurrencyCode);

/* Set the currency code using a JCHAR value /*

jdeMathSetCurrencyCodeUNI(&lpDS->mnAmount, (JCHAR *) szCurrencyCode);
```

サードパーティの API

サードパーティのプログラミング・インタフェース(API)には、Unicode 文字列をサポートしていないものがあります。このような場合は、文字列を非 Unicode フォーマットに変換してから API を呼び出し、その後は J.D. Edwards ERP 9.0 で保管できるように Unicode フォーマットに戻す必要があります。非 Unicode API をプログラミングする際には、次のガイドラインに従ってください。

- API の string 型パラメータごとに、Unicode 変数と非 Unicode 変数を宣言します。
- API を呼び出す前に、Unicode 文字列を非 Unicode 文字列に変換します。
- パラメータ・リストで非 Unicode 文字列を渡して API を呼び出します。
- 戻された非 Unicode 文字列を、J.D. Edwards ERP 9.0 で保管できるように Unicode 文字列に変換します。

例 : サードパーティの API

次の例では、サードパーティの API である GetStateName を呼び出しています。この GetStateName は、2 文字の状況コードを受け入れて 30 文字の状況名を戻します。

```
/* Declare variables */
JCHAR       szStateCode[3] = _J("CO"); /* Unicode state code */
JCHAR       szStateName[31] = _J("¥0");/* Unicode state name */
ZCHAR       zzStateCode[3] = _Z("¥0");  /* Non-Unicode state code */
ZCHAR       zzStateName[31] = _Z("¥0");          /* Non-Unicode state name */
BOOL        bReturnStatus = FALSE;               /* API return flag */
/* Convert unicode strings to non-unicode strings */
jdeFromUnicode(zzStateCode, szStateCode, DIM(zzStateCode), NULL);
/* Call API */
bReturnStatus = GetStateName(zzStateCode, zzStateName);
/* Convert non-unicode strings to unicode strings for storage in
 * OneWorld® */
jdeToUnicode(szStateName, zzStateName, DIM(szStateName), NULL);
```

フラット・ファイル API

jdeFprintf()など、ERP 9.0 の API はデータを変換します。これは、文字データ用のデフォルトのフラット・ファイル I/O が Unicode で行われることを意味します。ERP 9.0 で生成されるフラット・ファイルのユーザーは、Unicode に対応していなければ、そのフラット・ファイルを正しく読み取ることができません。そのため、追加の API セットを使用します。

対話型アプリケーションでは、ユーザーはアプリケーション名、アプリケーションのバージョン名、ユーザー名、および環境名などの属性に基づいて、フラット・ファイルのエンコーディングを構成できます。API セットには、ファイル I/O 関数である fwrite/fread、fprintf/fscanf、fputs/fgets および fputc/fgetc が組み込まれています。API は、構成アプリケーションで指定されたコード・ページを使用してデータを変換します。そのアプリケーションまたはバージョンの構成を変換関数で検出できるように、関数には追加のパラメータ lpBhvrCom を渡す必要があります。

これらの新規 API を呼び出す必要があるのは、ERP 9.0 外部のプロセスがフラット・ファイル・データを書き込んだり読み込む場合のみです。単なるワークテーブルやデバッグ・ファイルを ERP 9.0 で書き込んだり読み込む場合は、非変換 API（jdeFprintf()など）を使用します。

例:フラット・ファイル API

次の例では、ERP 9.0 で読み込まれるのみのフラット・ファイルにテキストを書き込んでいます。ファイルのエンコーディングは Unicode です。

```
FILE *fp;

fp  = jdeFopen(_J( "c:/testBSFNZ.txt"), _J("w+"));

jdeFprintf(fp, _J("%s%d¥n"), _J("Line "), 1);

jdeFclose(fp);
```

次の例では、サードパーティ・システムで読み込まれるフラット・ファイルにテキストを書き込んでいます。ファイルのエンコーディングは、構成済みのエンコーディングに基づきます。

```
FILE *fp;

fp  = jdeFopen(_J( "c:/testBSFNZ.txt"), _J("w+"));

jdeFprintfConvert(lpBhvrCom, fp, _J("%s%d¥n"), _J("Line "), 1);

jdeFclose(fp);
```

標準ヘッダー・ファイルおよびソース・ファイル

〈Business Function Design(ビジネス関数の設計)〉では、.c および.h テンプレートが生成されます。ここでは、テンプレートの各セクションの情報の詳細と関連情報について説明します。

ビジネス関数の.c および.h モジュールの両方について、モデルとなるソース・コード・ファイルが存在します。

標準ヘッダー

コンパイラがビジネス関数を正常に作成するには、ヘッダー・ファイルが必要です。C 言語には、33 のキーワードがあります。その他の printf や getchar はすべて関数です。各関数は、ビジネス関数の先頭に組み込むヘッダー・ファイルに定義されています。ヘッダー・ファイルがなければ、コンパイラでは関数が認識されず、エラー・メッセージが戻されます。

次の例に、ビジネス関数ソース・ファイルの標準ヘッダーを示します。

```
/*****************************************************************
 *   Header File:  BXXXXXXX.h
 *   Description:  Generic Business Function Header File
 *       History:
 *         Date        Programmer  SAR# - Description
 *         ----------  ----------  ------------------------------
 *  Author 06/06/2003                  - Created
 *
 * Copyright (c) J.D. Edwards World Source Company, 2003
 *
 * 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 */
```

ビジネス関数名と記述

Business Function Name および Description セクションを使用して、ビジネス関数の名称と記述を定義し、修正ログを管理します。

著作権情報

Copyright セクションには、J.D. Edwards & Company の著作権情報が含まれており、各ソース・ファイルに組み込む必要があります。このセクションは変更しないでください。

ビジネス関数のヘッダー定義

ビジネス関数のヘッダー定義セクションには、ビジネス関数の#define が含まれています。この部分はツールにより生成されます。このセクションは変更しないでください。

テーブル・ヘッダー組込項目

Table Header Inclusions セクションには、ビジネス関数が直接アクセスするテーブルに関連付けられたテーブル・ヘッダーが組み込まれます。

参照

- include ステートメントでの小文字

外部ビジネス関数ヘッダー組込項目

External Business Function Header Inclusions セクションには、外部で定義されたビジネス関数に関連付けられ、そのビジネス関数が直接アクセスするビジネス関数が含まれます。

参照

- include ステートメントでの小文字

グローバル定義

Global Definitions セクションを使用して、ビジネス関数が使用するグローバル定数を定義します。定数名には大文字を使用し、複数の場合はアンダースコアで区切って指定します。

参照

- define ステートメントの使用

構造体の定義

Structure Definitions セクションでは、ビジネス関数が使用する構造体を定義します。構造体名の前にソース・ファイル名を付けて、他のビジネス関数に含まれる同じ名前の構造体と競合しないようにする必要があります。

参照

- 命名規則
- typedef ステートメントの使用

DS テンプレート・タイプ定義

DS Template Type Definitions セクションは修正しないでください。このセクションでは、ヘッダーに対応するソースに含まれるビジネス関数が定義されます。構造体は、〈オブジェクト管理ワークベンチ〉の〈ビジネス関数設計〉または〈データ構造体設計〉ウィンドウで生成されます。

ソース・プリプロセッサの定義

Source Preprocessing Definitions セクションでは、ビジネス関数のエントリ・ポイントを定義し、C 関数で必要な左中カッコを組み込みます。このセクションは変更しないでください。

ビジネス関数プロトタイプ

Business Function Prototypes セクションを使用して、ソース・ファイルに定義されている関数をプロトタイプ化します。

参照

- 関数プロトタイプの作成

内部関数のプロトタイプ

Internal Function Prototypes セクションには、関数の記述とパラメータが含まれています。

参照

- 命名規則
- 関数プロトタイプの作成

標準ソース

ソース・ファイルには、ビジネス関数の指示が含まれています。ここでは、標準ソースの各セクションについて説明します。

この後のページに、J.D. Edwards ソフトウェアのビジネス関数の作成時に標準ソース・ファイル用に生成されるテンプレートを示します。

```
#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
* SourceCompany.
* 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>

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

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

/***************************************************************
 *  Business Function:  GenericBusinessFunction
 *
 *    Description:  Generic Business Function
 *
 *    Parameters:
 *    LPBHVRCOM    lpBhvrCom Business Function Communications
 *    LPVOID       lpVoid    Void Parameter - DO NOT USE!
 *    LPDSDXXXXXXX lpDS      Parameter Data Structure Pointer
 *
 ***************************************************************/
JDEBFRTN (ID) JDEBFWINAPI GenericBusinessFunction
              (LPBHVRCOM      lpBhvrCom,
               LPVOID         lpVoid,
               LPDSDXXXXXXXX  lpDS)
{
   /***************************************************************
    *  Variable declarations
    ***************************************************************/

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

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

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

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

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

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

   return (ER_SUCCESS);
}

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

```
*      Notes:
*
*    Returns:
*
* Parameters:
*************************************************************/
```

ビジネス関数名と記述

このセクションを使用して、ビジネス関数の名称と記述を管理します。また、このセクションは修正ログの管理にも使用します。

著作権情報

Copyright セクションには、J.D. Edwards & Company の著作権情報が含まれており、各ソース・ファイルに組み込む必要があります。このセクションは変更しないでください。

注記

Notes セクションには、今後コードを検討することになる担当者のための情報を組み込みます。たとえば、ビジネス関数や特別なロジックに関連する特性を記述します。

グローバル定義

Global Definitions セクションを使用して、ビジネス関数が使用するグローバル定数を定義します。

参照

- define ステートメントの使用

関連ビジネス関数のヘッダー・ファイル

Header File for Associated Business Function セクションでは、#include を使用してビジネス関数に関連するヘッダー・ファイルをインクルードします。ソースにヘッダー・ファイルを追加インクルードする必要がある場合は、それをここでインクルードします。

ビジネス関数ヘッダー

Business Function Header セクションには、ビジネス関数が使用する各パラメータの記述が含まれています。このセクションは変更しないでください。

変数宣言

Variable Declarations セクションでは、関数に必須の変数をすべて定義します。使用しやすいように、各変数をタイプ別に順次定義してください。

参照

- 命名規則
- 変数の初期化

構造体宣言

Declare Structures セクションでは、関数に必須の構造体をすべて定義します。

参照

- 変数名

ポインタ

関数にポインタが必要な場合は、この Pointers セクションで定義します。ポインタには、どの構造体を指しているかが認識しやすい名称を設定してください。たとえば、lpDS1100 は構造体 DS1100 を指しています。

参照

- 変数名

NULL ポインタのチェック

Check for NULL Pointers セクションでは、NULL であるパラメータ・ポインタがあるかどうかをチェックします。このセクションは変更しないでください。

ポインタの設定

Set Pointers セクションを使用するのは、変数を宣言時に初期化していない場合です。定義したポインタすべてに値を割り当てる必要があります。

参照

- 変数とデータ構造体の宣言と初期化

主要な処理機能

Main Processing セクションにはコードを記述します。

関数のクリーンアップ

Function Clean Up セクションを使用して、割当済みメモリを解放します。

参照

- 関数クリーンアップ領域の使用

内部関数コメント・ ブロック

Internal Function Comment Block セクションには、関数の記述とパラメータが含まれます。

参照

- 命名規則
- 関数プロトタイプの作成