JD EDWARDS ENTERPRISEONE

JD Edwards EnterpriseOne Tools 8.96 Development Tools: APIs and Business Functions Guide

April 2006

JD Edwards EnterpriseOne Tools 8.96 Development Tools: APIs and Business Functions Guide
SKU E1_TOOLS896TBF-B 0406

Contents

General Preface

About This Documentation Preface xi

JD Edwards EnterpriseOne Application Prerequisites....... xi

Application Fundamentals....... xi

Documentation Updates and Printed Documentation....... xii

  Obtaining Documentation Updates....... xii

  Ordering Printed Documentation....... xii

Additional Resources....... xiii

Typographical Conventions and Visual Cues....... xiv

  Typographical Conventions....... xiv

  Visual Cues....... xv

  Country, Region, and Industry Identifiers....... xv

  Currency Codes....... xvi

Comments and Suggestions....... xvi

Common Fields Used in Implementation Guides....... xvi

Preface

JD Edwards EnterpriseOne Tools: APIs and Business Functions Preface....... xix

JD Edwards EnterpriseOne Tools Fundamentals....... xix

Chapter 1

Getting Started with JD Edwards EnterpriseOne Tools: APIs and Business Functions....... 1

Development Tools: APIs and Business Functions Overview....... 1

Development Tools: APIs and Business Functions Implementation....... 1

  Development Tools: APIs and Business Functions Implementation Steps....... 1

Chapter 2

Working with APIs....... 3

Understanding APIs....... 3

  API Fundamentals....... 3

  Common Library APIs....... 3

  Database APIs....... 5

Calling APIs..........7
Calling an API from an External Business Function..........8
Calling a Visual Basic Program from JD Edwards EnterpriseOne Software..........9
Using the SAX Parser..........10
Understanding the SAX Parser..........10
Examples of SAX Parser Usage..........11
Example of a SAX Parsing Sequence..........18
Working with JDECACHE..........19
Understanding Caching..........19
Understanding the JDECACHE API set..........21
Understanding JDECACHE Standards..........22
Prerequisites..........23
Calling JDECACHE APIs..........23
Setting Up Indexes..........24
Initializing the Cache..........26
Using an Index to Access the Cache..........27
Using the jdeCacheInit/jdeCacheTerminate Rule..........28
Using the Same Cache in Multiple Business Functions or Forms..........28
Working with JDECACHE Cursors..........29
Opening a JDECACHE Cursor..........29
Using the JDECACHE data set..........30
Updating Records..........32
Deleting Records..........32
Using the jdeCacheFetchPosition API..........33
Using the jdeCacheFetchPositionByRef API..........33
Resetting the Cursor..........33
Closing the Cursor..........33
Using JDECACHE Multiple Cursor Support..........33
Using JDECACHE Partial Keys..........34

Chapter 3
Using Business Functions..........35
Understanding Business Functions..........35
Components of a Business Function..........36
How Distributed Business Functions Work..........38
C Business Functions..........39
Business Function Event Rules..........49
Understanding Transaction Master Business Functions..........51
Building Transaction Master Business Functions..........53

Understanding Building Transaction Master Business Functions....................54
Begin Document....................55
Edit Line....................58
Edit Document....................60
End Document....................61
Clear Cache....................62
Cancel Document....................63
Implementing Transaction Master Business Functions....................64
Single-Record Processing....................64
Document Processing....................65
Working with Master File Master Business Functions....................66
MBF Information Structure....................67
Master Business Function Impact on Performance....................69
Working with Business Functions....................69
Prerequisite....................70
Creating a Custom DLL....................70
Specifying a Custom DLL for a Custom Business Function....................70
Working with Business Function Builder....................70
Setting Build Options....................71
Reading Build Output....................71
Building All Business Functions....................73
Using the Utility Programs....................75
Understanding Business Function Processing Failovers....................84
Working with Business Function Documentation....................85
Understanding Business Function Documentation....................85
Creating Business Function Documentation....................85
Viewing Documentation from Business Function Documentation Viewer....................86

Chapter 4
Understanding Record Locking....................87
Record Locking....................87
Optimistic Locking....................87
Pessimistic Locking....................88
Using Pessimistic Locking Within a Transaction Boundary....................88
Business Functions and Pessimistic Locking....................88

Chapter 5
Debugging Business Functions........89
Debugging........89
Debugging Strategies........89
Debug Logs........90
Debugging Business Functions with Microsoft Visual C++........90
Understanding the Visual C++ Debugger........91
Understanding Visual C++ Debugger Tracing Utilities........92
Debugging Business Functions Attached to Interactive Applications........92
Using SQL Log Tracing........93
Using Debug Tracing........93

Appendix A
JD Edwards EnterpriseOne APIs........95
General APIs........95
jdeCreateGuid........95
jdeCreateGuidString........95
jdeGuidCompare........96
jdeGuidToString........97
jdeEncryptWKey........98
jdeDecryptWKey........100
JDB_TextSearchClearSelection........101
JDB_TextSearchClearSequencing........102
JDB_TextSearchCloseView........102
JDB_TextSearchFetch........103
JDB_TextSearchOpenView........104
JDB_TextSearchSelect........105
JDB_TextSearchSetSelection........106
JDB_TextSearchSetSequencing........107
TextSearchFullIndexing........109
TextSearchIncrementIndexing........110
TextSearchIndexClearing........111
TextSearchIndexOptimizing........112
Dynamic Logging APIs........113
chgOutputLoggingLevel........114
chgIPCTraceLevel........115
chgNetTraceLevel........116
chgTAMTraceLevel........117
chgCMTraceLevel........117

chgCMTraceFilter........119
chgPSThreadTraceLevel........119
chgSecTraceLevel........120
chgSaveEVNDoc........121
chgIEOTraceLevels........122
jdeCache APIs........124
jdeCacheAdd........124
jdeCacheClear........125
jdeCacheCloseCursor........126
jdeCacheDelete........127
jdeCacheDeleteAll........128
jdeCacheFetch........130
jdeCacheFetchPosition........132
jdeCacheFetchPositionByRef........134
jdeCacheGetIndex........135
jdeCacheGetNumCursors........137
jdeCacheGetNumRecords........138
jdeCacheInit........139
jdeCacheInitEx........141
jdeCacheInitMultipleIndex........143
jdeCacheInitMultipleindexEx........145
jdeCacheInitMultipleIndexUser........147
jdeCacheInitUser........149
jdeCacheOpenCursor........151
jdeCacheResetCursor........152
jdeCacheSetIndex........153
jdeCacheTerminate........155
jdeCacheTerminateAll........157
jdeCacheUpdate........158
JD Edwards EnterpriseOne Threadsafe APIs........160
jdePPSRand........160
jdePPRand........161
jdeLocaltime........162
Media Object APIs........162
jdeGT_CloseTable........162
jdeGT_DeleteData/jdeGT_DeleteDataKeyStr........163
jdeGT_FetchData/jdeGT_FetchDataEx........167
jdeGT_InsertData/jdeGT_InsertDataKeyStr........172
jdeGT_OpenTable........176
jdeGT_SelectData/jdeGT_SelectDataKeyStr........178

jdeGT_UpdateData/jdeGT_UpdateDataKeyStr........182
jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang........186
jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr........189
jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr........193
jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr........196
jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr........200
jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr........203
jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr........207
jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr........210
jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr........213
jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr........216
jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr........219
jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr........222
jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr........225
jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr........228
jdeGTDelete_HTML/jdeGTDelete_HTMLKeyStr........231
jdeGTDelete_Image/jdeGTDelete_ImageKeyStr........235
jdeGTDelete_OLE/jdeGTDelete_OLEKeyStr........238
jdeGTDelete_Shortcut/jdeGTDelete_ShortcutKeyStr........242
jdeGTDelete_Text/jdeGTDelete_TextKeyStr........245
jdeGTDelete_Vendor/jdeGTDelete_VendorKeyStr........249
jdeGTFreeMOData........252
jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr........255
jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr........258
jdeGTGet_HTML/jdeGTGet_HTMLKeyStr........262
jdeGTGet_Image/jdeGTGet_ImageKeyStr........266
jdeGTGet_OLE/jdeGTGet_OLEKeyStr........269
jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr........273
jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr........276
jdeGTGet_Vendor/jdeGTGet_VendorKeyStr........280
jdeGTGetCount/jdeGTGetCountKeyStr........284
jdeValidateGTExist/jdeValidateGTExistWithKeyStr........287
Messaging and Workflow APIs........291
DoSendMessagev3........291
SAX Interface Functions........294
Structure Used With SAX Parser Interface Functions........294
XRCS_initEngine........294
XRCS_getParserByType........294
XRCS_getParser (DOM only)........295

XRCS_setCallback........295
XRCS_setCallbackWithOption........296
XRCS_parseXMLFile........297
XRCS_parseXMLString........298
XRCS_freeParser........298
XRCS_terminateEngine........298
Callback Functions........299
Errors and Warnings........299
Callback Function Format 1........300
Callback Function Format 2........300
Callback Function Format 3........301
Callback Function Format 4........301
Callback Function Format 5........302

Glossary of JD Edwards EnterpriseOne Terms........303

Index........313

About This Documentation Preface

JD Edwards EnterpriseOne implementation guides provide you with the information that you need to implement and use JD Edwards EnterpriseOne applications from Oracle.

This preface discusses:

- JD Edwards EnterpriseOne application prerequisites.
- Application fundamentals.
- Documentation updates and printed documentation.
- Additional resources.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common fields in implementation guides.

Note. Implementation guides document only elements, such as fields and check boxes, that require additional explanation. If an element is not documented with the process or task in which it is used, then either it requires no additional explanation or it is documented with common fields for the section, chapter, implementation guide, or product line. Fields that are common to all JD Edwards EnterpriseOne applications are defined in this preface.

JD Edwards EnterpriseOne Application Prerequisites

To benefit fully from the information that is covered in these books, you should have a basic understanding of how to use JD Edwards EnterpriseOne applications.

You might also want to complete at least one introductory training course, if applicable.

You should be familiar with navigating the system and adding, updating, and deleting information by using JD Edwards EnterpriseOne menus, forms, or windows. You should also be comfortable using the World Wide Web and the Microsoft Windows or Windows NT graphical user interface.

These books do not review navigation and other basics. They present the information that you need to use the system and implement your JD Edwards EnterpriseOne applications most effectively.

Application Fundamentals

Each application implementation guide provides implementation and processing information for your JD Edwards EnterpriseOne applications.

For some applications, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals implementation guide. Most product lines have a version of the application fundamentals implementation guide. The preface of each implementation guide identifies the application fundamentals implementation guides that are associated with that implementation guide.

The application fundamentals implementation guide consists of important topics that apply to many or all JD Edwards EnterpriseOne applications. Whether you are implementing a single application, some combination of applications within the product line, or the entire product line, you should be familiar with the contents of the appropriate application fundamentals implementation guides. They provide the starting points for fundamental implementation tasks.

Documentation Updates and Printed Documentation

This section discusses how to:

- Obtain documentation updates.
- Order printed documentation.

Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on Oracle's PeopleSoft Customer Connection website. Through the Documentation section of Oracle's PeopleSoft Customer Connection, you can download files to add to your Implementation Guides Library. You'll find a variety of useful and timely materials, including updates to the full line of JD Edwards EnterpriseOne documentation that is delivered on your implementation guides CD-ROM.

Important! Before you upgrade, you must check Oracle's PeopleSoft Customer Connection for updates to the upgrade instructions. Oracle continually posts updates as the upgrade process is refined.

See Also

Oracle's PeopleSoft Customer Connection, http://www.oracle.com/support/support_peoplesoft.html

Ordering Printed Documentation

You can order printed, bound volumes of the complete line of JD Edwards EnterpriseOne documentation that is delivered on your implementation guide CD-ROM. Oracle makes printed documentation available for each major release of JD Edwards EnterpriseOne shortly after the software is shipped. Customers and partners can order this printed documentation by using any of these methods:

- Web
- Telephone
- Email

Web

From the Documentation section of Oracle's PeopleSoft Customer Connection website, access the PeopleBooks Press website under the Ordering PeopleBooks topic. Use a credit card, money order, cashier's check, or purchase order to place your order.

Telephone

Contact MMA Partners, the book print vendor, at 877 588 2525.

Email

Send email to MMA Partners at peoplebookspress@mmapartner.com.

See Also

Oracle's PeopleSoft Customer Connection, http://www.oracle.com/support/support_peoplesoft.html

Additional Resources

The following resources are located on Oracle's PeopleSoft Customer Connection website:

Resource	Navigation
Application maintenance information	Updates + Fixes
Business process diagrams	Support, Documentation, Business Process Maps
Interactive Services Repository	Support, Documentation, Interactive Services Repository
Hardware and software requirements	Implement, Optimize, and Upgrade; Implementation Guide; Implementation Documentation and Software; Hardware and Software Requirements
Installation guides	Implement, Optimize, and Upgrade; Implementation Guide; Implementation Documentation and Software; Installation Guides and Notes
Integration information	Implement, Optimize, and Upgrade; Implementation Guide; Implementation Documentation and Software; Pre-Built Integrations for PeopleSoft Enterprise and JD Edwards EnterpriseOne Applications
Minimum technical requirements (MTRs) (JD Edwards EnterpriseOne only)	Implement, Optimize, and Upgrade; Implementation Guide; Supported Platforms
Documentation updates	Support, Documentation, Documentation Updates
Implementation guides support policy	Support, Support Policy
Prerelease notes	Support, Documentation, Documentation Updates, Category, Release Notes
Product release roadmap	Support, Roadmaps + Schedules
Release notes	Support, Documentation, Documentation Updates, Category, Release Notes
Release value proposition	Support, Documentation, Documentation Updates, Category, Release Value Proposition
Statement of direction	Support, Documentation, Documentation Updates, Category, Statement of Direction

Resource	Navigation
Troubleshooting information	Support, Troubleshooting
Upgrade documentation	Support, Documentation, Upgrade Documentation and Scripts

Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions.
- Visual cues.
- Country, region, and industry identifiers.
- Currency codes.

Typographical Conventions

This table contains the typographical conventions that are used in implementation guides:

Typographical Convention or Visual Cue	Description
Bold	Indicates PeopleCode function names, business function names, event names, system function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Italics	Indicates field values, emphasis, and JD Edwards EnterpriseOne or other book-length publication titles. In PeopleCode syntax, italic items are placeholders for arguments that your program must supply. We also use italics when we refer to words as words or letters as letters, as in the following: Enter the letter *O*.
KEY+KEY	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press the W key.
`Monospace font`	Indicates a PeopleCode program or other code example.
“ ” (quotation marks)	Indicate chapter titles in cross-references and words that are used differently from their intended meanings.

Typographical Convention or Visual Cue	Description
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.

Visual Cues

Implementation guides contain the following visual cues.

Notes

Notes indicate information that you should pay particular attention to as you work with the JD Edwards EnterpriseOne system.

Note. Example of a note.

If the note is preceded by *Important!,* the note is crucial and includes information that concerns what you must do for the system to function properly.

Important! Example of an important note.

Warnings

Warnings indicate crucial configuration considerations. Pay close attention to warning messages.

Warning! Example of a warning.

Cross-References

Implementation guides provide cross-references either under the heading "See Also" or on a separate line preceded by the word *See.* Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.

Country, Region, and Industry Identifiers

Information that applies only to a specific country, region, or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a country-specific heading: “(FRA) Hiring an Employee”

Example of a region-specific heading: “(Latin America) Setting Up Depreciation”

Country Identifiers

Countries are identified with the International Organization for Standardization (ISO) country code.

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in implementation guides:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in implementation guides:

- USF (U.S. Federal)
- E&G (Education and Government)

Currency Codes

Monetary amounts are identified by the ISO currency code.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like to see changed about implementation guides and other Oracle reference and training materials. Please send your suggestions to Documentation Manager, Oracle Corporation, 7604 Technology Way, Denver, CO, 80237. Or email us at documentation_us@oracle.com.

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions.

Common Fields Used in Implementation Guides

Address Book Number Enter a unique number that identifies the master record for the entity. An address book number can be the identifier for a customer, supplier, company, employee, applicant, participant, tenant, location, and so on. Depending on the application, the field on the form might refer to the address book number as the customer number, supplier number, or company number, employee or applicant ID, participant number, and so on.

As If Currency Code

Enter the three-character code to specify the currency that you want to use to view transaction amounts. This code enables you to view the transaction amounts as if they were entered in the specified currency rather than the foreign or domestic currency that was used when the transaction was originally entered.

Batch Number

Displays a number that identifies a group of transactions to be processed by the system. On entry forms, you can assign the batch number or the system can assign it through the Next Numbers program (P0002).

Batch Date

Enter the date in which a batch is created. If you leave this field blank, the system supplies the system date as the batch date.

Batch Status

Displays a code from user-defined code (UDC) table 98/IC that indicates the posting status of a batch. Values are:

Blank: Batch is unposted and pending approval.

A: The batch is approved for posting, has no errors and is in balance, but has not yet been posted.

D: The batch posted successfully.

E: The batch is in error. You must correct the batch before it can post.

P: The system is in the process of posting the batch. The batch is unavailable until the posting process is complete. If errors occur during the post, the batch status changes to *E*.

U: The batch is temporarily unavailable because someone is working with it, or the batch appears to be in use because a power failure occurred while the batch was open.

Branch/Plant

Enter a code that identifies a separate entity as a warehouse location, job, project, work center, branch, or plant in which distribution and manufacturing activities occur. In some systems, this is called a business unit.

Business Unit

Enter the alphanumeric code that identifies a separate entity within a business for which you want to track costs. In some systems, this is called a branch/plant.

Category Code

Enter the code that represents a specific category code. Category codes are user-defined codes that you customize to handle the tracking and reporting requirements of your organization.

Company

Enter a code that identifies a specific organization, fund, or other reporting entity. The company code must already exist in the F0010 table and must identify a reporting entity that has a complete balance sheet.

Currency Code

Enter the three-character code that represents the currency of the transaction. JD Edwards EnterpriseOne provides currency codes that are recognized by the International Organization for Standardization (ISO). The system stores currency codes in the F0013 table.

Document Company

Enter the company number associated with the document. This number, used in conjunction with the document number, document type, and general ledger date, uniquely identifies an original document.

If you assign next numbers by company and fiscal year, the system uses the document company to retrieve the correct next number for that company.

If two or more original documents have the same document number and document type, you can use the document company to display the document that you want.

Document Number

Displays a number that identifies the original document, which can be a voucher, invoice, journal entry, or time sheet, and so on. On entry forms, you can assign the original document number or the system can assign it through the Next Numbers program.

Document Type

Enter the two-character UDC, from UDC table 00/DT, that identifies the origin and purpose of the transaction, such as a voucher, invoice, journal entry, or time sheet. JD Edwards EnterpriseOne reserves these prefixes for the document types indicated:

P: Accounts payable documents.

R: Accounts receivable documents.

T: Time and pay documents.

I: Inventory documents.

O: Purchase order documents.

S: Sales order documents.

Effective Date

Enter the date on which an address, item, transaction, or record becomes active. The meaning of this field differs, depending on the program. For example, the effective date can represent any of these dates:

- The date on which a change of address becomes effective.
- The date on which a lease becomes effective.
- The date on which a price becomes effective.
- The date on which the currency exchange rate becomes effective.
- The date on which a tax rate becomes effective.

Fiscal Period and **Fiscal Year**

Enter a number that identifies the general ledger period and year. For many programs, you can leave these fields blank to use the current fiscal period and year defined in the Company Names & Number program (P0010).

G/L Date (general ledger date)

Enter the date that identifies the financial period to which a transaction will be posted. The system compares the date that you enter on the transaction to the fiscal date pattern assigned to the company to retrieve the appropriate fiscal period number and year, as well as to perform date validations.

JD Edwards EnterpriseOne Tools: APIs and Business Functions Preface

This preface discusses Oracle's JD Edwards EnterpriseOne Tools 8.96 Guide: APIs and Business Functions.

JD Edwards EnterpriseOne Tools Fundamentals

Additional information describing the setup and design of the JD Edwards EnterpriseOne Tools system resides in companion documentation that apply to many or all JD Edwards EnterpriseOne Tools. These companion guides contain information that applies specifically to JD Edwards EnterpriseOne Tools APIs and Business Functions.

- *JD Edwards EnterpriseOne Tools 8.96 PeopleBook: System Administration*
- *JD Edwards EnterpriseOne Tools 8.96 PeopleBook: Object Management Workbench*

See Also

JD Edwards EnterpriseOne Tools 8.96 System Administration Guide, "Getting Started with JD Edwards EnterpriseOne Tools System Administration"

JD Edwards EnterpriseOne Tools 8.96 Object Management Workbench Guide, "Getting Started with JD Edwards EnterpriseOne OMW"

CHAPTER 1

Getting Started with JD Edwards EnterpriseOne Tools: APIs and Business Functions

This chapter discusses:

- Development Tools: APIs and Business Functions Overview
- Development Tools: APIs and Business Functions Implementation

Development Tools: APIs and Business Functions Overview

Development Tools: APIs and Business Functions is used to create complex, reusable routines in C. Business functions can call APIs directly, and can in turn be invoked from event rules (ER).

Development Tools: APIs and Business Functions Implementation

This section provides an overview of the steps that are required to implement Development Tools: APIs and Business Functions.

In the planning phase of the implementation, take advantage of all JD Edwards sources of information, including the installation guides and troubleshooting information. A complete list of these resources appears in the preface in *About This Documentation* with information about where to find the most current version of each.

Development Tools: APIs and Business Functions Implementation Steps

This table lists the steps for the Development Tools: APIs and Business Functions implementation.

Step	Reference
1. Configure Object Management Workbench.	*JD Edwards EnterpriseOne Tools 8.96 Object Management Workbench Guide*, “Configuring JD Edwards EnterpriseOne OMW”
2. Configure Object Management Workbench user roles and allowed actions.	*JD Edwards EnterpriseOne Tools 8.96 Object Management Workbench Guide*, “Configuring User Roles and Allowed Actions”

Step	Reference
3. Configure Object Management Workbench functions.	*JD Edwards EnterpriseOne Tools 8.96 Object Management Workbench Guide*, "Configuring JD Edwards EnterpriseOne OMW Functions"
4. Configure Object Management Workbench activity rules.	*JD Edwards EnterpriseOne Tools 8.96 Object Management Workbench Guide*, "Configuring Activity Rules"
5. Configure Object Management Workbench save locations.	*JD Edwards EnterpriseOne Tools 8.96 Object Management Workbench Guide*, "Configuring Object Save Locations"
6. Set up default location and printers.	*JD Edwards EnterpriseOne Tools 8.96 Development Tools: Report Printing Administration Technologies Guide*, "Getting Started with JD Edwards EnterpriseOne Report Printing Administration Technologies"

CHAPTER 2

Working with APIs

This chapter provides an overview of Oracle's JD Edwards EnterpriseOne APIs and discusses how to:

- Call APIs.
- Use the SAX parser.
- Work with JDECACHE.
- Work with JDECACHE cursors.

Understanding APIs

This section discusses:

- API fundamentals
- Common library APIs
- Database APIs

API Fundamentals

APIs are routines that perform predefined tasks. JD Edwards EnterpriseOne APIs make it easier for third-party applications to interact with JD Edwards EnterpriseOne software. These APIs are functions that you can use to manipulate JD Edwards EnterpriseOne data types, provide common functionality, and access the database. Several categories of APIs exist, including the Common Library Routines and JD Edwards EnterpriseOne Database (JDEBASE) APIs.

Programing with APIs is useful for these reasons:

- No code modifications are required as functionality is upgraded.
- When a data structure changes, source modifications are minimal to nonexistent.
- Common functionality is provided through the APIs, and they are less prone to error.

When the code in an API changes, business functions typically only need to be recompiled and relinked.

Common Library APIs

The Common Library APIs, such as determining whether foreign currency is enabled, manipulating the date format, retrieving link list information, or retrieving math numeric and date information, are specific to JD Edwards EnterpriseOne functionality. You can use these APIs to set up data by calling APIs and modifying data after API calls. Some of the more commonly used categories of APIs include MATH_NUMERIC, JDEDATE, and LINKLIST. Other miscellaneous Common Library APIs are also available.

JD Edwards EnterpriseOne provides the data types, MATH_NUMERIC and JDEDATE, for use when creating business functions. Because these data types might change, you must use the Common Library APIs provided by JD Edwards EnterpriseOne to manipulate the variables of these data types.

MATH_NUMERIC Data Type

The MATH_NUMERIC data type exclusively represents all numeric values in JD Edwards EnterpriseOne software. The values of all numeric fields on a form or batch process are communicated to business functions in the form of pointers to MATH_NUMERIC data structures. MATH_NUMERIC is used as a data dictionary (DD) data type.

The data type is defined as follows:

```
struct tagMATH_NUMERIC
{
 ZCHAR String[MAXLEN_MATH_NUMERIC+1];/* Just the digits - no separators */
 BYTE Sign;    /* - if negative, 0x00 otherwise */
 ZCHAR EditCode;   /* The Data Dictionary edit code to Format for display */
 short nDecimalPosition;  /* # of digits from right end of string to decimal point⇒
 */
 short nLength;   /* The number of digits in s */
 WORD wFlags;   /* Processing Flags */
 ZCHAR szCurrency[CURRENCY_CODE_SIZE];/* The Currency Code */
 short nCurrencyDecimals;  /* The Number of Currency Decimals */
 short nPrecision;   /* The Data Dictionary Size */
};
```

This table lists various elements:

MATH_NUMERIC Element	Description
String	Digits without separators
Sign	A minus sign indicates the number is negative, otherwise the value is 0x00
EditCode	Data dictionary edit code that formats the number for display
nDecimalPosition	Number of digits from the right to place the decimal
nLength	Number of digits in the string
wFlags	Processing flags
szCurrency	Currency code
nCurrencyDecimals	Number of currency decimals
nPrecision	Data dictionary size

JDEDATE Data Type

The JDEDATE data type exclusively represents all dates in JD Edwards EnterpriseOne software. The values of all date fields on a form or batch process are communicated to business functions in the form of pointers to JDEDATE data structures. JDEDATE is used as a data dictionary data type.

This code sample illustrates defining the data type:

```
struct tagJDEDATE
{
 short nYear;;
 short nMonth;;
 short nDay;
};
typedef struct tagJDEDATE JDEDATE, FAR *LPJDEDATE;
```

This table lists the elements in the JDEDATE data type:

JDEDATE Element	Description
nYear	Year (4 digits)
nMonth	Month
nDay	Day

Database APIs

JD Edwards EnterpriseOne software supports multiple databases. An application can access data from a number of databases.

Standards and Portability

These standards affect the development of relational databases:

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

Ideally, industry standards enable users to work identically with different relational database systems. Although each major vendor supports industry standards, it also offers extensions to enhance the functionality of the SQL language. Vendors also periodically release 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 that is not affected by differences between database vendors. When a vendor provides a new release, the affect on existing applications should be minimal. To solve many of these portability issues, many organizations use standard database interfaces called open database connectivity (ODBC).

JD Edwards EnterpriseOne ODBC

JD Edwards EnterpriseOne ODBC enables you to use one set of functions to access multiple relational database management systems. Consequently, you can develop and compile applications knowing that they can run on a variety of database types with the correct database driver. Database drivers are installed that enable the JD Edwards EnterpriseOne ODBC interface to communicate with a specific database system using a database driver.

The driver handles the I/O buffers to the database, which enables a programmer to write an application that communicates with a generic data source. The database driver is responsible for processing the API request and communicating with the correct data source. The application does not have to be recompiled to work with other databases. If the application must perform the same operation with another database, a new driver is loaded.

A driver manager handles all application requests to the JD Edwards EnterpriseOne database function call. The driver manager processes the request or passes it to an appropriate driver.

JD Edwards EnterpriseOne applications access data from heterogeneous databases, using the JDB API to interface between the applications and multiple databases. Applications and business functions use the JDB API to dynamically generate platform-specific SQL statements. JDB also supports additional features, such as replication and cross-data source joins.

Standard JDEBASE API Categories

You can use control and request level APIs to develop and test business functions. This table lists the categories of JDEBASE APIs:

Category	Description
Control Level	Provides functions for initializing and terminating the database connection.
Request Level	Provides functions for performing database transactions. The request level functions perform these tasks: • Connect to and disconnect from tables and business views in the database. • Perform data manipulation operations of select, insert, update, and delete. • Retrieve data with fetch commands.
Column Level	Performs and modifies information for columns and tables.
Global Table/Column Specifications	Provides the capability to create and manipulate column specifications.

Connecting to a Database

To perform a request, the driver manager and driver must manage the information for the development environment, each application connection, and the SQL statement. The pointers that return this information to the application are called handles. The APIs must include these handles in each function call. Handles used by the development environment include these handles:

Handle	Purpose
HENV	The environment handle contains information related to the current database connection and valid connection handles. Every application connecting to the database must have an environment handle. This handle is required to connect to a data source.
HUSER	The user handle contains information related to a specific connection. Each user handle has an associated environment handle with it. A connection handle is required to connect to a data source. If you are using transaction processing, initializing HUSER indicates the beginning of a transaction.
HREQUEST	The request handle contains information related to a specific request to a data source. An application must have a request handle before executing SQL statements. Each request handle is associated with a user handle.

Understanding Database Communication Steps

Several APIs called in succession can perform these steps for database communication:

- Initialize communication with the database.
- Establish a connection to the specific data to access.
- Execute statements on the database.
- Release the connection to the database.
- Terminate communication with the database.

This table lists some of the API levels and the communication handles and API names that are associated with them:

API Level	Communication Handles	API Name
Control level (application or test driver)	Environment handle	**JDB_InitEnv**
Control level (application or test driver)	User handle (created)	**JDB_InitUser**
Request level (business function)	User handle (retrieved)	**JDB_InitBhvr**
Request level (business function)	Request handle	**JDB_OpenTable**
Request level (business function)	Request handle	**JDB_FetchKeyed()**
Request level (business function)	Request handle	**JDB_CloseTable**
Request level (business function)	User handle	**JDB_FreeBhvr**
Control level (application or test driver)	User handle	**JDB_FreeUser**
Control level (application or test driver)	Environment handle	**JDB_FreeEnv**

Calling APIs

This section discusses how to:

- Call an API from an external business function.
- Call a Visual Basic program from JD Edwards EnterpriseOne software.

Calling an API from an External Business Function

You can call APIs from external business functions. To call an API from an external business function, you must first determine the function-calling convention of the .dll that you are going to use. It can be either cdecl or stdcall. The code might change slightly depending on the calling convention. This information should be included in the documentation for the .dll. If you do not know the calling convention of the .dll, you can execute the *dumpbin* command to determine the calling convention. Execute this command from the MSDOS prompt window:

```
dumpbin /EXPORTS ExternalDll.DLL.
```

Dumpbin displays information about the dll. If the output contains function names preceded by _ and followed by an @ sign with additional digits, the dll uses the stdcall calling convention; otherwise, it uses cdecl.

Stdcall Calling Convention

This example is standard code for Windows programs and is not specific to JD Edwards EnterpriseOne software:

```
# ifdef JDENV_PC
HINSTANCE hLibrary = LoadLibrary(_TEXT(YOUR_LIBRARY.DLL)); // substitute the name⇒
 of the external dll
if(hLibrary)
{
// create a typedef for the function pointer based on the parameters and return⇒
 type of the function to be called. This information can be obtained
// from the header file of the external dll. The name of the function to be called⇒
 in the following code is StartInstallEngine. We create a typedef for
// a function pointer named PFNSTARTINSTALLENGINE. Its return type is BOOL. Its⇒
 parameters are HUSER, LPCTSTR, LPCTSTR, LPTSTR & LPTSTR.
// Substitute these with parameter and return types for the particular API.
typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR, LPTSTR);
// Now create a variable for the function pointer of the type you just created.⇒
 Then make call to GetProcAddress function with the first
// parameter as the handle to the library you just loaded. The second parameter⇒
 should be the name of the function you want to call prepended
// with an _, and appended with an @ followed by the total number of bytes for the⇒
 parameters. In this example, the total number of bytes in the
// parameters for StartInstallEngine is 20 ( 4 bytes for each parameter ). The Get⇒
ProcAddress API will return a pointer to the function that you need to
// call.
PFNSTARTINSTALLENGINE lpfnStartInstallEngine = (PFNSTARTINSTALLENGINE) GetProc⇒
Address(hLibrary, _StartInstallEngine@20);
if ( lpfnStartInstallEngine )
{
// Now call the API by passing in the requisite parameters.
lpfnStartInstallEngine(hUser, szObjectName, szVersionName, pszObjectText, szObject⇒
Type);
}
#endif
```


Cdecl Calling Convention

The process for using the cdecl calling convention is similar to the process for using the std calling convention. They differ principally in the second parameter for **GetProcAddress**. Note the comments that precede that call.

```
# ifdef JDENV_PC
HINSTANCE hLibrary = LoadLibrary(_TEXT(YOUR_LIBRARY.DLL)); // substitute the name⇒
 of the external dll
if(hLibrary)
{
// create a typedef for the function pointer based on the parameters and return⇒
 type of the function to be called. This information can be obtained
// from the header file of the external dll. The name of the function to be called⇒
 in the following code is StartInstallEngine. We create a typedef for
// a function pointer named PFNSTARTINSTALLENGINE. Its return type is BOOL. Its⇒
 parameters are HUSER, LPCTSTR, LPCTSTR, LPTSTR & LPTSTR.
// Substitute these with parameter and return types for the particular API.
typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR, LPTSTR);
// Now create a variable for the function pointer of the type you just created.⇒
 Then make call to GetProcAddress function with the first
// parameter as the handle to the library you just loaded. The second parameter⇒
 should be the name of the function you want to call. In this
// case it will be StartInstallEngine only. The GetProcAddress API will return a⇒
 pointer to the function that you need to call.
PFNSTARTINSTALLENGINE lpfnStartInstallEngine = (PFNSTARTINSTALLENGINE) GetProc⇒
Address(hLibrary, StartInstallEngine);
if ( lpfnStartInstallEngine )
{
// Now call the API by passing in the requisite parameters.
lpfnStartInstallEngine(hUser, szObjectName, szVersionName, pszObjectText, szObject⇒
Type);
}
#endif
```

Note. These calls work only on a Windows *client* machine. **LoadLibrary** and **GetProcAddress** are Windows APIs. If the business function is compiled on a *server*, the compile will fail.

Calling a Visual Basic Program from JD Edwards EnterpriseOne Software

You can call a Visual Basic program from a JD Edwards EnterpriseOne business function and pass a parameter from the Visual Basic program to the JD Edwards EnterpriseOne business function using this process:

1. Write the Visual Basic program into a Visual Basic .dll that exports the function name of the program and returns a parameter to the JD Edwards EnterpriseOne business function.
2. Write a business function that loads the Visual Basic .dll using the win32 function LoadLibrary.
3. In the business function that you create, call the win32 function GetProcAddress to get the Visual Basic function and call it.

Using the SAX Parser

This section provides an overview of the SAX parser and of examples for its use.

Understanding the SAX Parser

The SAX parser is one of two main parsers used for XML data. It is an events-based parser, as opposed to the other XML parser, DOM, which is a tree-based parser. The Xerces product, from the Apache organization, provides both XML parsers. The Xerces code is written in C++. To make XML parsing available to business functions, a C-API interface, XercesWrapper, exists to provide access to both parsers. The design of the parsers is quite different, and that provides advantages for each parser, depending on the intended usage.

The DOM parser reads the XML file and builds an internal model (DOM document tree) of that file in memory. This has the advantage of enabling you to traverse the tree, retrieve parent-child relationships, and revisit the same data multiple times. The disadvantages include high memory requirements for large XML files. Also, the entire XML file must be read into memory before any of the data in the DOM document tree can begin to be processed. The DOM parser can also be used to programmatically build a DOM document tree in memory, and then write that tree to a file, in XML format.

The SAX parser reads an XML file and as each item is read, the parser passes that piece of data to callback functions. This methodology has the advantage of enabling fast processing with minimal memory usage. Also, the parsing can be stopped after a specific item has been found. The disadvantages include that the current state of parsing must be maintained by the callback functions, and previous data items can not be revisited without rereading the XML file. Finally, the SAX parser is a read-only parser.

This is a typical sequence used for parsing an XML data file using the DOM parser:

1. Initialize the XercesWrapper, which in turn, initializes the Xerces code.
2. Initialize the DOM parser.
3. Parse the XML data file.
4. Retrieve a pointer to the root element of the DOM document tree.
5. Retrieve additional elements and data, by traversing the DOM document tree.

   The callback functions are called whenever the specified events in the XML file are parsed.
6. Free all DOM elements that have been retrieved.
7. Free the DOM document tree.
8. Free the DOM parser.
9. Terminate the XercesWrapper interface, which in turn, closes the Xerces code.

This is a typical sequence used for parsing an XML data file, using the SAX parser:

1. Initialize the XercesWrapper, which in turn, initializes the Xerces code.
2. Initialize the SAX parser.
3. Set up various callback functions for specific parsing events.
4. Parse the XML data file.
5. Call the callback functions as each event in the XML file is parsed.
6. Within the callback functions, process the retrieved data and maintain a context for coordination between callback functions.

7. Free the SAX parser.
8. Terminate the XercesWrapper interface, which in turn, closes the Xerces code.

Examples of SAX Parser Usage

Many of the initialization, parsing, and termination functions are the same for both SAX and DOM parsers. The major difference is that the DOM parser returns a document handle which is then used with the traversing and data retrieval functions. Those functions are not used with SAX. SAX does all of the data processing within the user-defined callback functions. The callback functions are not used with DOM.

The processing of SAX-parsed data items occurs within the callback functions. Typically, each callback function maintains a context. The context can be passed to all callback functions and can be implemented as a data structure. The context, plus the other data passed to the callback functions, enables each data item to be processed appropriately.

Example Context Data Structure

This is a sample function which uses the SAX parser:

```
typedef struct tagParserCallbackValues {
 FILE *fp;
 JCHAR *szIndentString;
 int nIndentLevel;
} ZCALLBACK_VALUES, *PCALLBACK_VALUES;
```

Example Main Function

This is a sample context data structure:

```
/* SAX callbacks - display callback events into file */
int testcase_read_15(JCHAR *m_infile, JCHAR *m_outfile)
{
 XRCS_Status XRCSStatus;
 XRCS_hParser hParser;
 ZCALLBACK_VALUES zCbValues;
 PCALLBACK_VALUES pCbValues = &zCbValues;

 /* initialize context structure */
 pCbValues->fp = NULL;
 pCbValues->szIndentString = _J(" ");
 pCbValues->nIndentLevel = 0;

 /* open display file */
 pCbValues->fp = jdeFopen(m_outfile, _J("w"));

 if (pCbValues->fp != NULL)
 {
  XRCSStatus = XRCS_initEngine();
  if(XRCSStatus != XRCS_SUCCESS) {
   return -1;
  }
```

```
XRCSStatus = XRCS_getParserByType(&hParser, XRCS_SAX_PARSER_TYPE);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_START_DOC,
 (void *) cb_startDoc_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

/* set up callbacks for the SAX parser */
XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_END_DOC,
 (void *) cb_endDoc_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_START_ELEM,
 (void *) cb_startElement_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_END_ELEM,
 (void *) cb_endElement_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_CHARACTERS,
 (void *) cb_characters_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

XRCSStatus = XRCS_setCallback(hParser,
 XRCS_CALLBACK_IGNORABLE_WHITESPACE,
 (void *) cb_ignorableWhitespace_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}

XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_FATAL_ERROR,
 (void *) cb_fatalError_Display, (void *) pCbValues);
if(XRCSStatus != XRCS_SUCCESS) {
 return -1;
}
```



```
    XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_ERROR,
     (void *) cb_error_Display, (void *) pCbValues);
    if(XRCSStatus != XRCS_SUCCESS) {
     return -1;
    }

    XRCSStatus = XRCS_setCallback(hParser, XRCS_CALLBACK_WARNING,
     (void *) cb_warning_Display, (void *) pCbValues);
    if(XRCSStatus != XRCS_SUCCESS) {
     return -1;
    }

    /* now do the actual parsing */
    XRCSStatus = XRCS_parseXMLFile(hParser,m_infile, NULL);
    if(XRCSStatus != XRCS_SUCCESS) {
     return -1;
    }

    XRCSStatus = XRCS_freeParser(hParser);
    XRCSStatus = XRCS_terminateEngine();

    /* close display file */
    jdeFclose(pCbValues->fp);
   }
   else
   {
    /* could not open display file */
    return -1; }

   return 0;
  }
```

Example Callback Functions

These are sample callback functions:

```
/* callbacks for display of SAX parser events */
XRCS_CallbackStatus cb_startDoc_Display(void *pContext)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;

 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("START DOCUMENT"));
 return( XRCS_CB_CONTINUE);
}

XRCS_CallbackStatus cb_endDoc_Display(void *pContext)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;
```

```
 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("END DOCUMENT"));
 indentNewLine(pCbValues);
 return( XRCS_CB_CONTINUE);
}

XRCS_CallbackStatus cb_startElement_Display(void *pContext,
 const JCHAR *szUri,
 const JCHAR *szLocalname,
 const JCHAR *szQname,
 unsigned int nNumAttrs,
 const XRCS_ATTR_INFO *pAttributes)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;
 unsigned int nAttrNum;
 const XRCS_ATTR_INFO * thisAttr = NULL;

 pCbValues->nIndentLevel++;
 /* display element name */
 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("ELEMENT: "));
 if (jdeStrlen( szLocalname) != 0)
 {
  jdeFprintf(pCbValues->fp, _J("<%ls"), szLocalname);
 }
 else
 {
  jdeFprintf(pCbValues->fp, _J("<%ls"), szQname);
 }
 /* display attributes */
 if (nNumAttrs > 0U)
 {
  for (nAttrNum = 0U; nAttrNum < nNumAttrs; nAttrNum++)
  {
   thisAttr = &pAttributes[nAttrNum];
   /* display attrribute name */
   indentNewLine(pCbValues);
   jdeFprintf(pCbValues->fp, _J(" ATTR: "));
   if (jdeStrlen( thisAttr->szAttrLocalname) != 0)
   {
   jdeFprintf(pCbValues->fp, _J("%ls"),
    thisAttr->szAttrLocalname);
   }
   else
   {
   jdeFprintf(pCbValues->fp, _J("%ls"), thisAttr->szAttrQname);
   }
   /* display attribute value */
   jdeFprintf(pCbValues->fp, _J(" \""));
   jdeFprintf(pCbValues->fp, _J("%ls"), thisAttr->szAttrValue);
```



```
      jdeFprintf(pCbValues->fp, _J("\""));
     }
    indentNewLine(pCbValues);
   }
  /* display close of element name */
  jdeFprintf(pCbValues->fp, _J(">"));
  return( XRCS_CB_CONTINUE);
 }

XRCS_CallbackStatus cb_endElement_Display_Terminate(void *pContext,
 const JCHAR *szUri,
 const JCHAR *szLocalname,
 const JCHAR *szQname)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;

 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("END_ELM: "));
 if (jdeStrlen( szLocalname) != 0)
 {
  jdeFprintf(pCbValues->fp, _J("</%ls>"), szLocalname);
 }
 else
 {
  jdeFprintf(pCbValues->fp, _J("</%ls>"), szQname);
 }
 pCbValues->nIndentLevel--;
 return( XRCS_CB_TERMINATE);
}

XRCS_CallbackStatus cb_endElement_Display(void *pContext,
 const JCHAR *szUri,
 const JCHAR *szLocalname,
 const JCHAR *szQname)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;

 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("END_ELM: "));
 if (jdeStrlen( szLocalname) != 0)
 {
  jdeFprintf(pCbValues->fp, _J("</%ls>"), szLocalname);
 }
 else
 {
  jdeFprintf(pCbValues->fp, _J("</%ls>"), szQname);
 }
 pCbValues->nIndentLevel--;
 return( XRCS_CB_CONTINUE);
}
```

```
XRCS_CallbackStatus cb_warning_Display(void *pContext,
 XRCS_CallbackType eCallbackType,
 int nLineNum,
 int nColNum,
 const JCHAR *szPublicId,
 const JCHAR *szSystemId,
 const JCHAR *szMessage)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;

 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("Warning: "));
 jdeFprintf(pCbValues->fp, _J(" %ls (%ls) - %ls found at Column %d
 Line %d"), szSystemId, szPublicId, szMessage, nColNum, nLineNum);
 return( XRCS_CB_CONTINUE);
}

XRCS_CallbackStatus cb_error_Display(void *pContext,
 XRCS_CallbackType eCallbackType,
 int nLineNum,
 int nColNum,
 const JCHAR *szPublicId,
 const JCHAR *szSystemId,
 const JCHAR *szMessage)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;

 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("Error: "));
 jdeFprintf(pCbValues->fp, _J(" %ls (%ls) - %ls found at Column %d
 Line %d"), szSystemId, szPublicId, szMessage, nColNum, nLineNum);
 return( XRCS_CB_CONTINUE);
}

XRCS_CallbackStatus cb_fatalError_Display(void *pContext,
 XRCS_CallbackType eCallbackType,
 int nLineNum,
 int nColNum,
 const JCHAR *szPublicId,
 const JCHAR *szSystemId,
 const JCHAR *szMessage)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;

 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("Fatal Error: "));
 jdeFprintf(pCbValues->fp, _J(" %ls (%ls) - %ls found at Column %d Line %d"),
 szSystemId, szPublicId, szMessage, nColNum, nLineNum);
 return( XRCS_CB_TERMINATE);
```

```
}

XRCS_CallbackStatus cb_characters_Display(void *pContext,
 const JCHAR *szText)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;
 int nTextLen;
 int nTextRemaining;
 int nTextPieceLen;
 int nTextStartPosition;

 nTextLen = jdeStrlen( szText);
 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("CHARS: "));
 if (hasPrintingChars( szText, nTextLen) == TRUE)
 {
  /* initial quote */
  jdeFprintf(pCbValues->fp, _J("\""), szText);
  /* actual text, output in blocks of 10000 characters */
  /* jdeFprintf will not work with very large strings */
  nTextRemaining = nTextLen;
  nTextStartPosition = 0;
  while (nTextRemaining > 0)
  {
   if (nTextRemaining > 10000)
   {
   nTextPieceLen = 10000;
   }
   else
   {
   nTextPieceLen = nTextRemaining;
   }
   jdeFprintf(pCbValues->fp, _J("%.*ls"), nTextPieceLen,
   (JCHAR *) &(szText[nTextStartPosition]));
   nTextRemaining -= nTextPieceLen;
   nTextStartPosition += nTextPieceLen;
  }
  /* trailing quote */
  jdeFprintf(pCbValues->fp, _J("\""), szText);
 }
 return( XRCS_CB_CONTINUE);
}

XRCS_CallbackStatus cb_ignorableWhitespace_Display(void *pContext,
 const JCHAR *szText)
{
 PCALLBACK_VALUES pCbValues = (PCALLBACK_VALUES) pContext;
 int nTextLen;

 nTextLen = jdeStrlen( szText);
```



```
 indentNewLine(pCbValues);
 jdeFprintf(pCbValues->fp, _J("IGNORABLE WHITESPACE: "));
 if (hasPrintingChars( szText, nTextLen) == TRUE)
 {
  jdeFprintf(pCbValues->fp, _J("\"%ls\""), szText);
 }
 return( XRCS_CB_CONTINUE);
}

void indentNewLine(PCALLBACK_VALUES pCbValues)
{
 int nIndent = 0;

 jdeFprintf(pCbValues->fp,
  _J("\n"));

 while (nIndent < pCbValues->nIndentLevel)
 {
  jdeFprintf(pCbValues->fp, _J("%ls"), pCbValues->szIndentString);
  nIndent++;
 }
}

BOOL hasPrintingChars( const JCHAR *szText, int nTextLen)
{
 BOOL bHasPrinting = FALSE;
 int nText = 0;

 /* true if contains any printing characters */
 /* false if all blanks or control characters */
 while (nText < nTextLen)
 {
  if (szText[nText] > _J(' '))
  {
   bHasPrinting = TRUE;
   break;
  }
  nText++;
 }
 return( bHasPrinting);
}
```

Example of a SAX Parsing Sequence

This is an example of the sequence of callback functions called, for an example string of XML data. Before parsing, these callback functions were set up:

- **cb_startAllElements** for start-of-element event type.
- **cb_endAllElements** for end-of-element event type.

- **cb_startElement1** for start-of-element, with optional name specified as "elapsedTime."
- **cb_endElement1** for end-of-element, with optional name specified as "elapsedTime."
- **cb_chars** for characters event type.
- **cb_allCharacters** for characters, with optional setting for characters after elements.
- **cb_fatalError** for fatal-error event type.

The example XML string to be parsed is:

```
<main>startMain<elapsedTime>123</elapsedTime>endMain</main>
```

This callback sequence results from parsing this XML string:

- **cb_startAllElements** for *main*.
- **cb_chars** for *startMain*.
- **cb_allCharacters** for *startMain*.
- **cb_startAllElements** for *elapsedTime*.
- **cb_startElement1** for *elapsedTime*.
- **cb_chars** for *123*.
- **cb_allCharacters** for *123*.
- **cb_endAllElements** for *elapsedTime*.
- **cb_endElement1** for *elapsedTime*.
- **cb_allCharacters** for *endMain*.
- **cb_endAllElements** for *main*.
- **cb_fatalError** is not called while parsing this example XML string.

Working with JDECACHE

This section provides overviews of caching, JDECACHE standards, and the JDECACHE API set, and discusses how to:

- Call JDECACHE APIs.
- Set up indices.
- Initialize the cache.
- Use an index to access the cache.
- Use the jdeCacheInit/jdeCacheTerminate rule.
- Use the same cache in multiple business functions or forms.

Understanding Caching

Caching is a process that stores a local copy of frequently accessed content of remote objects. Caching can improve performance. JD Edwards EnterpriseOne software caches information in these ways:

- The system automatically caches some tables, such as those associated with constants, when it reads them from the database at startup.

  It caches these tables to a user's workstation or to a server for faster data access and retrieval.

- Individual applications can be enabled to use cache.

  JDECACHE APIs enable the server or workstation memory to be used as temporary storage.

JDECACHE is a component of JDEKRNL that can hold any type of indexed data that the application needs to store in memory, regardless of the platform on which the application is running; therefore, an entire table can be read from a database and stored in memory. No limitations exist regarding the type of data, size of data, or number of data caches that an application can have, other than the limitations of the computer on which it is running. Both fixed-length and variable-length records are supported. To use JDECACHE on any supported platform, you need to know only a simple set of API calls.

Data handled by JDECACHE is in RAM. Therefore, ensure that you really need to use JDECACHE. If you use JDECACHE, design the records and indices carefully. Minimize the number of records that you store in JDECACHE because JD Edwards EnterpriseOne software and various other applications need this memory as well.

JDECACHE supports multiple cursors, multiple indexes, and partial keys processing. JDECACHE is flexible in terms of positioning within the cache for data manipulation, which improves performance by reducing searching within the cache.

The JDB environment creates, manages, and destroys the JDECACHE environment. Each cache that you use within the JDECACHE environment is associated with a JDB user. Therefore, you must call JDB_InitBhvr API before you call any of the JDECACHE APIs.

When to Use JDECACHE

Here is a scenario that highlights when an application might use the JDECACHE APIs.

You use workfiles when an application must store records that a user enters in a detail area until OK processing is activated upon the *Button Clicked* event. On OK processing, all records must be simultaneously updated to the database. This is similar to transaction processing. For example, in the detail area of purchase order detail, if a user enters 30 lines of information and then decides to cancel the transaction, all records in the workfile are deleted and nothing is written to the database. As the user exits each detail row, editing takes place for each field, and then that record is written to the workfile.

If you implement this situation without using workfiles, irreversible updates to database tables occur when the user exits each row. Using workfiles enables you to limit updates to tables so that they only occur on OK button processing, and they are included in a transaction boundary. The workfile defines a data boundary for the grid for processing purposes. This is useful when multiple applications or processes (such as business functions) must access the data in the workfile for updates and calculations.

Using cache might increase performance in some cases. You can use JDECACHE to store in memory the records that the user enters in one purchase order. The number of records that you store depends on the cache buffer size for each record, the local memory size, the location in which the business function that you use runs (for example, server or workstation), and so on. Typically, you should not store more than 1000 records. For example, do not cache the entire Address Book table in memory.

Performance Considerations

Follow these guidelines to get the best JDECACHE performance:

- Cache as few records as possible.
- The fewer columns (segments) that you use, the faster the search, insert, and delete actions occur.

In some cases, the system might have to compare each column before it determines whether to go further in the cache.

- The fewer records in the cache, the faster all operations proceed.

Understanding the JDECACHE API set

You use a set of public APIs to interact with JDECACHE. You must understand how the JDECACHE APIs are organized to implement them effectively.

JDECACHE Management APIs

You can manage cache using the JDECACHE management APIs for these purposes:

- Setting up the cache.
- Clearing the cache.
- Terminating the cache.

Use the **jdeCacheGetNumRecords** and **jdeCacheGetNumCursors** APIs to retrieve cache statistics. They are only passed the HCACHE handle. All other JDECACHE management APIs should always be passed these handles:

- HUSER
- HCACHE

These two handles are essential for cache identification and cache management.

The set of JDECACHE management APIs consist of these APIs:

- **jdeCacheInit**
- **jdeCacheInitEx**
- **jdeCacheInitMultipleIndex**
- **jdeCacheInitMultipleIndexEx**
- **jdeCacheInitUser**
- **jdeCacheInitMultipleIndexUser**
- **jdeCacheGetNumRecords**
- **jdeCacheGetNumCursors**
- **jdeCacheClear**
- **jdeCacheTerminate**
- **jdeCacheTerminateAll**

The **jdeCacheInit** and **jdeCacheInitMultipleIndex** APIs initialize the cache uniquely per user. Therefore, if a user logs in to the software and then runs two sessions of the same application simultaneously, the two application sessions will share the same cache. Consequently, if the first application deletes a record from the cache, the second application cannot access the record. Conversely, if two users log in to the software and then run the same application simultaneously, the two application sessions have different caches. Consequently, if the first application deletes a record from its cache, the second application will still be able to access the record in its own cache. The **jdeCacheInitEx** and **jdeCacheInitMultipleIndexEx** APIs function exactly the same, but they additionally enable you to define the maximum number of cursors that can be opened by the cache.

The **jdeCacheInitUser** and **jdeCacheInitMultipleIndexUser** APIs initialize the cache uniquely per application. Therefore, if a user logs in to the software and then runs two sessions of the same application simultaneously, the two application sessions will have different caches. Consequently, if the first application deletes a record from its cache, the second application can still access the record in its own cache.

JDECACHE Manipulation APIs

You can use the JDECACHE manipulation APIs for retrieving and manipulating the data in the cache. Each API implements a cursor that acts as pointer to a record that is currently being manipulated. This cursor is essential for navigation within the cache. JDECACHE manipulation APIs should be passed handles of these types:

- HCACHE

  Identifies the cache that is being worked.

- HJDECURSOR

  Identifies the position in the cache that is being worked.

The set of JDECACHE manipulation APIs contain these APIs:

- **jdeCacheOpenCursor**
- **jdeCacheResetCursor**
- **jdeCacheAdd**
- **jdeCacheFetch**
- **jdeCacheFetchPosition**
- **jdeCacheUpdate**
- **jdeCacheDelete**
- **jdeCacheDeleteAll**
- **jdeCacheCloseCursor**
- **jdeCacheFetchPositionByRef**
- **jdeCacheSetIndex**
- **jdeCacheGetIndex**

Understanding JDECACHE Standards

It is recommended that you apply several standards when using JDECACHE. This section discusses the standards for business functions and programming.

The cache business function name should follow the standard naming convention for business functions.

Cache Business Function Source Description

These standards apply to source descriptions for cache business functions:

- The cache business function description must follow the business function description standards.
- The first word must be the noun, *Cache*.
- The second word must be the verb, *Process*.

- For an individual cache function, the words following *Process* should describe the cache. For a common cache function, the words following *Process* should describe the group to which the individual cache functions belong.

These standards apply to cache business function descriptions:

- If the source file contains an individual function, the function name must match the source name.
- If the source file contains a group of cache functions, the individual function names must follow the same standards as the Cache Business Function Source Description standards.

Cache Programming Standards

A variety of cache programming standards apply:

- General standards.
- Cache termination instead of clearing.
- Cache name.
- Cache data structure definition.
- Data structure standard data items.
- Cache action code standards.
- Group cache business function header file.
- Individual cache business function header file.

Prerequisites

Before you can use JDECACHE, you must:

- Define an index

  The index specifies to the cache the fields in a record that are used to uniquely identify a cache record.
- Initialize a cache

  Each group of data that an index references requires a separate cache.

Calling JDECACHE APIs

JDECHACHE APIs must be called in a certain order. This list defines the order in which the JDECACHE-related APIs must be called:

1. Call **JDB_InitBhvr**.
2. Create index or indices.
3. Call **jdeCacheInit**, **jdeCacheInitEx**, **jdeCacheInitMultipleIndex**, or **jdeCacheInitMultipleIndexEx**.
4. Call **jdeCacheAdd**.
5. Call **jdeCacheOpenCursor**.
6. Call JDECACHE Operations.

At JDECACHE Operations, the actual JDECACHE APIs can be called in any order. The operations in this list of JDECACHE operations can occur in any order:

- **jdeCacheFetch**

- **jdeCacheOpenCursor** (the second cursor)
- **jdeCacheFetchPosition**
- **jdeCacheUpdate**
- **jdeCacheDelete**
- **jdeCacheDeleteAll**
- **jdeCacheResetCursor**
- **jdeCacheCloseCursor** (if the second cursor is opened)
- **jdeCacheCloseCursor**
- **jdeCacheTerminate**
- **JDB_FreeBhvr**

Setting Up Indexes

To store or retrieve any data in JDECACHE, you must set up at least one index that consists of at least one column. The index is limited to a maximum of 25 columns (which are called segments) in the index structure. Use the data type provided to tell the cache manager what the index looks like. You must provide the number of columns (segments) in the index and the offset and size of each column in the data structure. To maximize performance, minimize the number of segments.

This code is the definition of the structure that holds index information:

```
#define JDECM_MAX_UM_SEGMENTS 25
struct _JDECMKeySegment
{
 short int nOffset;  /* Offset from beginning of structure in bytes */
 short int nSize;  /* Size of data item in bytes */
 int idDataType;  /* EVDT_MATH_NUMERIC or EVDT_STRING*/
} JDECMKEYSEGMENT;
struct _JDECMKeyStruct
{
 short int nNumSegments;
 JDECMKEYSEGMENT CacheKey[JDECM_MAX_NUM_SEGMENTS];
} JDECMINDEXSTRUCT;
```

Observe these rules when you create indices in JDECACHE:

- Always declare the index structure as an array that holds one element for single indexes.

  Declare the index structure as an array that holds more than one element for multiple indexes. You can create an unlimited number of indexes.
- Always use `memset()` for the index structure.

  When you use `memset()` for multiple indexes, multiply the size of the index structure by the total number of indexes.
- Always assign as elements the number of segments that correspond to the number of columns that you have in the CacheKey array.
- Always use `offsetof ()` to indicate the offset of a column in the structure that contains the columns.

This example illustrates a single index with multiple fields:


```
/* Example of single index with multiple fields.*/
JDECMINDEXSTRUCT Index[1]    = {0};
memset(&dsCache,0x00,sizeof(dsCache));
/* Initialize cache. */
Index->nNumSegments=5;
Index->CacheKey[0].nOffset=offsetof(DSCACHE,szEdiUserId);
Index->CacheKey[0].nSize=DIM(dsCache.szEdiUserId);
Index->CacheKey[0].idDataType=EVDT_STRING;
Index->CacheKey[1].nOffset=offsetof(DSCACHE,szEdiBatchNumber);
Index->CacheKey[1].nSize=DIM(dsCache.szEdiBatchNumber);
Index->CacheKey[1].idDataType=EVDT_STRING;
Index->CacheKey[2].nOffset=offsetof(DSCACHE,szEdiTransactNumber);
Index->CacheKey[2].nSize=DIM(dsCache.szEdiTransactNumber);
Index->CacheKey[2].idDataType=EVDT_STRING;
Index->CacheKey[3].nOffset=offsetof(DSCACHE,mnEdiLineNumber);
Index->CacheKey[3].nSize=sizeof(dsCache.mnEdiLineNumber);
Index->CacheKey[3].idDataType=EVDT_MATH_NUMERIC;
Index->CacheKey[4].nOffset=offsetof(DSCACHE.cErrorCode);
Index->CacheKey[4].nSize = 1;
Index->CacheKey[4].idDataType=EVDT_CHAR
```

The flag, *idDataType*, indicates the data type of the particular key.

This example illustrates a cache with multiple indices and multiple fields:

```
Memset(jdecmIndex,0x00,sizeof(JDECMINDEXSTRUCT)*2);
jdecmIndex[0].nKeyID=1;
jdecmIndex[0].nNumSegments=6;
jdecmIndex[0].CacheKey[0].nOffset=offsetof(I1000042,szCostCenter);
jdecmIndex[0].CacheKey[0].nSize=DIM(dsI1000042.szCostCenter);
jdecmIndex[0].CacheKey[0].idDataType=EVDT_STRING;
jdecmIndex[0].CacheKey[1].nOffset=offsetof(I1000042,szObjectAccount);
jdecmIndex[0].CacheKey[1].nSize=DIM(dsI1000042.szObjectAccount);
jdecmIndex[0].CacheKey[1].idDataType=EVDT_STRING;
jdecmIndex[0].CacheKey[2].nOffset=offsetof(I1000042,szSubsidiary);
jdecmIndex[0].CacheKey[2].nSize=DIM(dsI1000042.szSubsidiary);
jdecmIndex[0].CacheKey[2].idDataType=EVDT_STRING;
jdecmIndex[0].CacheKey[3].nOffset=offsetof(I1000042,szSubledger);
jdecmIndex[0].CacheKey[3].nSize=DIM(dsI1000042.szSubledger);
jdecmIndex[0].CacheKey[3].idDataType=EVDT_STRING;
jdecmIndex[0].CacheKey[4].nOffset=offsetof(I1000042,szSubledgerType);
jdecmIndex[0].CacheKey[4].nSize=1;
jdecmIndex[0].CacheKey[4].idDataType=EVDT_STRING;
jdecmIndex[0].CacheKey[5].nOffset=offsetof(I1000042,szCurrencyCodeFrom);
jdecmIndex[0].CacheKey[5].nSize=DIM(dsI1000042.szCurrencyCodeFrom);
jdecmIndex[0].CacheKey[5].idDataType=EVDT_STRING;
************************ KEY 2 *******************************
jdecmIndex[1].nKeyID=2;
jdecmIndex[1].nNumSegments=7;
jdecmIndex[1].CacheKey[0].nOffset=offsetof(I1000042,szEliminationGroup);
jdecmIndex[1].CacheKey[0].nSize=DIM(dsI1000042.szEliminationGroup);
```

```
jdecmIndex[1].CacheKey[0].idDataType=EVDT_STRING;
jdecmIndex[1].CacheKey[1].nOffset=offsetof(I1000042,szCostCenter);
jdecmIndex[1].CacheKey[1].nSize=DIM(dsI1000042.szCostCenter);
jdecmIndex[1].CacheKey[1].idDataType=EVDT_STRING;
jdecmIndex[1].CacheKey[2].nOffset=offsetof(I1000042,szObjectAccout);
jdecmIndex[1].CacheKey[2].nSize=DIM(dsI1000042.szObjectAccount);
jdecmIndex[0].CacheKey[2].idDataType=EVDT_STRING;
jdecmIndex[1].CacheKey[3].nOffset=offsetof(I1000042,szSubsidiary);
jdecmIndex[1].CacheKey[3].nSize=DIM(dsI1000042.szSubsidiary);
jdecmIndex[1].CacheKey[3].idDataType=EVDT_STRING;
jdecmIndex[1].CacheKey[4].nOffset=offsetof(I1000042,szSubledger);
jdecmIndex[1].CacheKey[4].nSize=DIM(dsI1000042.szSubledger);
jdecmIndex[1].CacheKey[4].idDataType=EVDT_STRING;
jdecmIndex[1].CacheKey[5].nOffset=offsetof(I1000042,szSubledgerType);
jdecmIndex[1].CacheKey[5].nSize=1;
jdecmIndex[1].CacheKey[5].idDataType=EVDT_STRING;
jdecmIndex[1].CacheKey[6].nOffset=offsetof(I1000042,szCurrencyCodeFrom);
jdecmIndex[0].CacheKey[6].nSize=DIM(dsI1000042.szCurrencyCodeFrom);
jdecmIndex[0].CacheKey[6].idDataType=EVDT_STRING;
```

Initializing the Cache

After you set up the index or indices, call **jdeCacheInit**, **jdeCacheInitEx**, **jdeCacheInitMultipleIndex**, or **jdeCacheInitMultipleIndexEx**. to initialize (create) the cache. Pass a unique cache name so that JDECACHE can identify the cache. Pass the index to this API so that the JDECACHE knows how to reference the data that will be stored in the cache. Because each cache must be associated with a user, you must also pass the user handle obtained from the call to **JDB_InitUser**. This API returns an HCACHE handle to the cache that JDECACHE creates. This handle appears in every subsequent JDECACHE API to identify the cache.

The keys in the index must be identical for every **jdeCacheInit**, **jdeCacheInitEx**, **jdeCacheInitMultipleIndex**, and **jdeCacheInitMultipleIndexEx** call for that cache until it is terminated. The keys in the index must correspond in number, order, and type for that index each time that it is used.

After the cache has been initialized successfully, JDECACHE operations can take place using the JDECACHE APIs. The cache handle obtained from **jdeCacheInit** or **jdeCacheInitEx** must be passed for every JDECACHE operation. JDECACHE makes an internal Index Definition Structure that accesses the cache when it is populated.

Example: Index Definition Structure

In this scenario, assume that each record that the cache stores has this structure:

```
int nlnt1
JCHAR cLetter1
JCHAR cLetter2
JCHAR cLetter3
JCHAR szArray(5)
```

The next step is to determine which values to use to index each record in the cache uniquely. In this example, assume that these values are required:

- `nInt1`
- `cLetter1`

- `cLetter3`

Pass that information to **jdeCacheInit** or **jdeCacheInitEx**, and JDECACHE creates this Index Definition Structure for internal use. This table lists Index Definition Structure is for STRUCT letters:

Index Key No. Index Key #1	Index Key Offset 0	Index Key Offset INTEGER
Index Key #2	4	JCHAR
Index Key #3	6	JCHAR

Example of an index definition structure

Using an Index to Access the Cache

When you use an index to access the cache, the keys in the index that are sent to the API must correspond to the keys of the index used in the call to **jdeCacheInit** or **jdeCacheInitEx** for that cache in number, order, offset positions, and type. Therefore, if a field that was used in the index passed to **jdeCacheInit** or **jdeCacheInitEx** offsets position 99, it must also offset position 99 in the index structure that passed to JDECACHE access API.

You should use the same index structure that was used for the call to **jdeCacheInit** or **jdeCacheInitEx** whenever you call an API that requires an index structure.

The next example illustrates why the index offsets must be specified for the **jdeCacheInit** or **jdeCacheInitEx** and how they are used when a record is to be retrieved from the cache. It describes how the passed key is used in conjunction with the JDECACHE internal index definition structure to access cache records.

Example: JDECACHE Internal Index Definition Structure

In this example, assume that the user is looking for a record that matches these index key values:

- 1
- c
- i

JDECACHE accesses the values that you pass in the structure at the byte offsets that were defined in the call to **jdeCacheInit** or **jdeCacheInitEx**.

JDECACHE compares the values 1, c, and i that it retrieves from the passed structure to the corresponding values in each of the cache records at the corresponding byte offset. The cache records are stored as the structures that were inserted into the cache by **jdeCacheAdd**, which is the same structure as the one you pass first. The structure that matches the passed key is the second structure to which HCUR1 points.

You should never create a smaller structure that contains just the key to access the cache. Unlike most indexing systems, JDECACHE does not store a cache record's index separately from the actual cache record. This is because JDECACHE deals with memory-resident data and is designed to be as memory-conservative as possible. Therefore, JDECACHE does not waste memory by storing an extra structure for the sole purpose of indexing. Instead, a JDECACHE record has a dual purpose of index storage and data storage. This means that, when you retrieve a record from JDECACHE using a key, the key should be contained in a structure that is of the same type as the structure that is used to store the record in the cache.

Do not use any key structure to access the cache other than the one for which offsets that were defined in the index passed to **jdeCacheInit** or **jdeCacheInitEx**. The structure that contains the keys when accessing a cache should be the same structure that is used to store the cache records.

If **jdeCacheInit** or **jdeCacheInitEx** is called twice with the same cache name and the same user handle without an intermediate call to **jdeCacheTerminate**, the cache that was initialized using the first **jdeCacheInit** or **jdeCacheInitEx** will be retained. Always call **jdeCacheInit** or **jdeCacheInitEx** with the same index each time that you call it with the same cache name. If you call **jdeCacheInit** or **jdeCacheInitEx** for the same cache with a different index, none of the JDECACHE APIs will work.

The key for searches must always use the same structure type that stores cache records.

Using the jdeCacheInit/jdeCacheTerminate Rule

For every **jdeCacheInit**, **jdeCacheInitEx**, **jdeCacheInitMultipleIndex**, or **jdeCacheInitMultipleIndexEx**, a corresponding **jdeCacheTerminate** must exist, except instances in which the same cache is used across business functions or forms. In this case, all unterminated **jdeCacheInit**, **jdeCacheInitEx**, **jdeCacheInitMultipleIndex**, or **jdeCacheInitMultipleIndexEx** calls must be terminated with a **jdeCacheTerminateAll**.

A **jdeCacheTerminate** call terminates the most recent corresponding **jdeCacheInit** or **jdeCacheInitEx**. This means that the same cache can be used in nested business functions. In each function, perform a **jdeCacheInit** or **jdeCacheInitEx** or **jdeCacheInitEx** that passes the cache name. Before exiting that function, call **jdeCacheTerminate**. This does not destroy the cache. Instead, it destroys the association between the cache and the passed HCACHE handle. The cache is completely destroyed from memory only when the number of **jdeCacheTerminate** calls matches the number of **jdeCacheInit** or **jdeCacheInitEx** calls. In contrast, one call to **jdeCacheTerminateAll** destroys the cache from memory regardless of the number of **jdeCacheInit**, **jdeCacheInitEx**, **jdeCacheInitMultipleIndex**, or **jdeCacheInitMultipleIndexEx** calls or **jdeCacheTerminate** calls.

Using the Same Cache in Multiple Business Functions or Forms

If the same cache is required for two or more business functions or forms, call **jdeCacheInit** or **jdeCacheInitEx** in the first business function or form, and add data to it. After exiting that business function or form, do not call **jdeCacheTerminate** because this removes the cache from memory. Instead, in the subsequent business functions or forms, call **jdeCacheInit** or **jdeCacheInitEx** again with the same index and cache name as in the initial call to **jdeCacheInit** or **jdeCacheInitEx**. Because the cache was not terminated the first time, JDECACHE looks for a cache with the same name and assigns that to you. Because the cache already has records in it, you do not need to refresh it. You can proceed with normal cache operations on that cache.

If a cache is initialized multiple times across business functions or forms, use **jdeCacheTerminateAll** to terminate all instances of the cache that were initialized. The name of the cache that corresponds to the HCACHE passed to this API will be used to determine the cache to destroy. Use this API when you do not want to call **jdeCacheTerminate** for the number of times that **jdeCacheInit** or **jdeCacheInitEx** was called. If you move from one form or business function to another when you initialize the same cache across business functions or forms, you will lose the HCACHE because it is a local variable. To share the same cache across business functions or forms, do not call **jdeCacheTerminate** when you exit a form or business function if you intend to use the same cache in another form or business function.

Working with JDECACHE Cursors

JDECACHE Cursors (JDECACHE Cursor Manager) is a component of JDECACHE that implements a JDECACHE cursor for record retrieval and update. A JDECACHE cursor is a pointer to a record in a user's cache. The record after the record in which the cursor is currently pointing is the next record that will be retrieved from the cache upon calling a cache fetch API.

This section discusses how to:

- Open a JDECACHE cursor.
- Use the JDECACHE data set.
- Update records.
- Delete records.
- Use the **jdeCacheFetchPosition** API.
- Use the **jdeCacheFetchPostionByRef** API.
- Reset the cursor.
- Close the cursor.
- Use JDECACHE multiple cursor support.
- Use JDECACHE partial keys.

Opening a JDECACHE Cursor

Manipulating the JDECACHE data is cursor-dependent. Before the JDECACHE data manipulation APIs will work, a cursor must be opened. A cursor must be opened to obtain a cursor handle of the type HJDECURSOR, which must, in turn, be passed to all of the JDECACHE data manipulation APIs (with the exception of the **jdeCacheAdd** API). HJDECURSOR is the data type for the cursor handle. It must be passed to every API for JDECACHE data manipulation except **jdeCacheAdd**.

To open the cursor, call the **jdeCacheOpenCursor** API. A call to this API also makes possible the calls to all the data manipulation APIs (except for **jdeCacheAdd**). If you do not open the cursor, these APIs will *not* work. With this call, the cursor opens a JDECACHE data set, within which it will work. This API opens the data set, but does not fetch any data. This means that the cache must be initialized by a call to **jdeCacheInit** or **jdeCacheInitEx** and populated by a call to **jdeCacheAdd** before a cursor can be opened.

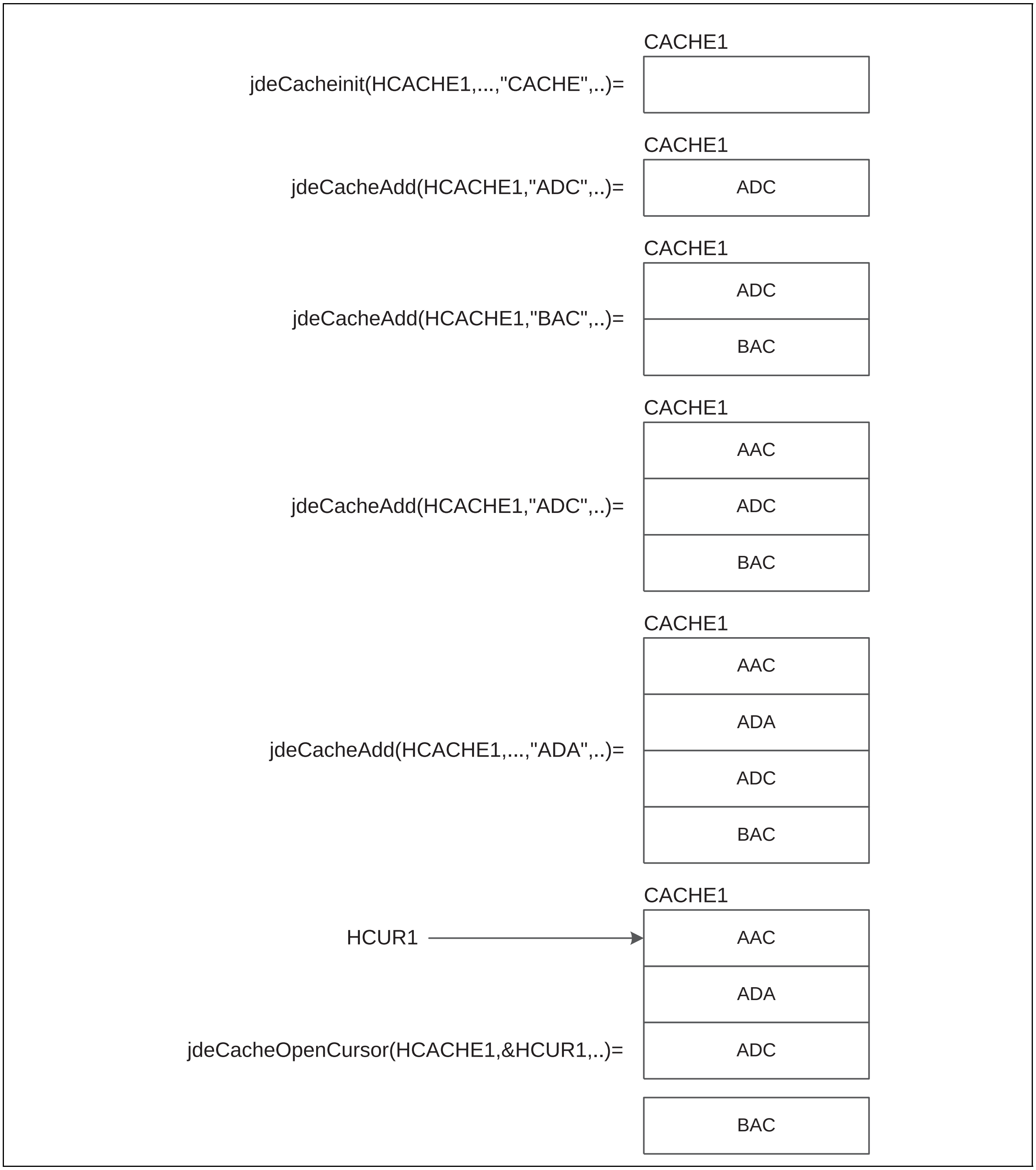
You can obtain multiple cursors to a cache by calling **jdeCacheOpenCursor** and passing different HJDECURSOR handles. In a multiple cursor environment, all the cursors are independent of each other.

When you are finished working with the cursor, you must deactivate it or close it by calling the **jdeCacheCloseCursor** API, and passing an HJDECURSOR handle that corresponds to the HJDECURSOR handle that was passed to the **jdeCacheOpenCursor**. When a cursor is closed, it cannot be used again until it is opened by a call to **jdeCacheOpenCursor**.

Using the JDECACHE data set

The JDECACHE data set includes all of the records from the current position of the cursor to the end of the set of sequenced records. Thus, if a cursor is in the middle of the data set, none of the records in the cache prior to the current position of the cursor is considered part of the data set. The JDECACHE data set consists of the cache records sequenced in ascending order of the given index keys. This means that the order in which the records have been placed in JDECACHE is not necessarily the order in which JDECACHE Cursors retrieves them. JDECACHE Cursors retrieves records in a sequential ascending order of the index keys. A forward movement by the cursor reduces the size of the data set during sequential retrievals. When the cursor advances past the last record in the data set, a failure is returned.

This example illustrates the creation of a JDECACHE cache and a JDECACHE data set:



Example of JDECACHE cache and data set creation

Cursor-Advancing APIs

Cursor-advancing JDECACHE fetch APIs implement the fundamental concepts of a cursor. The cursor-advancing API set consists of APIs that advance the cursor to the next record in the JDECACHE data set before fetching a record from JDECACHE. **jdeCacheFetch** and **jdeCacheFetchPosition** are examples of cursor-advancing fetch APIs.

A call to **jdeCacheFetch** first positions the cursor at the next record in the JDECACHE data set before retrieving it. JDECACHE Cursors also enable calls to position the cursor at a specific record within the data set. To do this, you call the **jdeCacheFetchPosition** API, which advances the cursor to the record that matches the given key before retrieving it.

You can use a combination of cursor-advancing fetch APIs if you need a sequential fetch of records starting from a certain position. Call **jdeCacheFetchPosition**, passing the key of the record from which you want to start retrieving. This advances the cursor to the desired location in the data set and retrieves the record. All subsequent calls to **jdeCacheFetch** will fetch records starting from the current cursor position in the data set until the end of the data set, or until the program stops for another reason.

Non-Cursor-Advancing APIs

Non-cursor-advancing JDECACHE cursor APIs do not advance the cursor before retrieving a record. Instead, they keep the cursor pointing to the retrieved record. **jdeCacheUpdate** and **jdeCacheDelete** are examples of non-cursor-advancing fetch APIs.

Updating Records

If you want to update a specific record with a key that you know, call **jdeCacheFetchPosition**, passing the known key, to position the cursor at the location of the record that matches the key. Because the cursor is already pointing to the desired location, call **jdeCacheUpdate**, passing the same HJDECURSOR that you used in the call to **jdeCacheFetchPosition**.

If the index key changes, cache re-sorts the records, and the cursor points to the updated location. However, when you call **jdeCacheFetch**, the system retrieves the next record in the updated set. Consequently, the system might not retrieve the correct record because the changed index key caused the order of the records to change.

To update a sequential number of records, make a call to **jdeCacheFetchPosition** to return to the beginning of the sequence, if necessary. Then call **jdeCacheUpdate**, passing the same HJDECURSOR that you used in the call to **jdeCacheFetchPosition**. This call updates only the record to which the cursor is pointing. To update the rest of the records in the sequence, call **jdeCacheFetch** repeatedly, passing the same HJDECURSOR that you used in the call to **jdeCacheFetchPosition**, until you get to the end of the sequence. A sequential update will not work correctly if you have changed any index key value. However, a sequential update will work correctly if you are updating a value that is not an index key.

Deleting Records

If you want to delete a specific record with a known key, first call **jdeCacheFetchPosition** to point the cursor to the location of the record that matches the key. Next, call **jdeCacheDelete**, to remove the record from cache. Pass **jdeCacheDelete** the same HJDECURSOR that you used when you called **jdeCacheFetchPosition**. After deleting a record, use **jdeCacheFetch** to retrieve the record that followed the now-deleted record. This process works only when you call **jdeCacheDelete**.

You can also delete a specific record by calling **jdeCacheDeleteAll** and passing it the full key with the specific record to be deleted. In this case, **jdeCacheFetch** will not work following **jdeCacheDeleteAll**, although you can work around this condition with **jdeCacheFetchPosition** or **jdeCacheResetCursor**.

To delete a sequential set of records, first call **jdeCacheFetchPosition** to point the cursor to the first record in the set or call **jdeCacheDeleteAll** to delete the first record in the set. Then, call **jdeCacheDelete** sequentially. In this case, **jdeCacheFetch** will not work following **jdeCacheDeleteAll**, although you can work around this condition with **jdeCacheFetchPosition** or **jdeCacheResetCursor**.

If you want to delete records that match a partial key, call **jdeCacheDeleteAll** and pass it a partial key. The system deletes all of the records that match the partial key. After you call this API, **jdeCacheFetch** does not work.

Using the jdeCacheFetchPosition API

The **jdeCacheFetchPosition** API searches for a specific record in the data set; therefore, it requires a specific key. This API can perform full and partial key searches.

Note. If you pass 0 for the number of keys, the system assumes that you want to perform a full key search.

Using the jdeCacheFetchPositionByRef API

The **jdeCacheFetchPositionByRef** API returns the address of a data set. The API finds the one record in cache and returns a reference (pointer) to the data. **jdeCacheFetchPositionByRef** retrieves a single, large block of data that is stored in cache. If the cache is empty or has more than one record, this API fails.

Resetting the Cursor

JDECACHE cursors supports multiple cursors, as well as an unlimited number of cursor oscillations within the data set. This means that the cursor can shuttle from beginning to end for an unlimited number of times. The cursor moves forward only. To reset the cursor (move the cursor back to the beginning of the data set), you must make a call to the **jdeCacheResetCursor** API to get a fresh JDECACHE data set.

You can also reset a cursor to a specific position that is outside of the current data set by calling the **jdeCacheFetchPosition** API.

Closing the Cursor

When you no longer need the cursor, call **jdeCacheCloseCursor** to close it. This call closes both the data set and the cursor. Any subsequent call to any JDECACHE API passing the closed HJDECURSOR without having called **jdeCacheOpenCursor** will fail.

Although opening a JDECACHE Cursor for a long period of time requires no overhead, to release the memory that it requires, you should close the cursor as soon as you no longer need it.

Using JDECACHE Multiple Cursor Support

JDECACHE supports multiple open cursors. Each cache that you initialize with **jdeCacheInit** or **jdeCacheInitMultipleIndex** enables up to 100 open cursors to access it at the same time. When you initialize a cache with **jdeCacheInitEx** or **jdeCacheInitMultipleIndexEx**, you can enable any number of cursors, between one and 100, to access it at the same time.

JDECACHE multiple cursors are designed to enable two or more asynchronously processing business functions to use one cache. Asynchronously processing business functions can open cursors to access the cache with relative positions within the cache that are independent of each other. A cursor movement by one business function does not affect any other open cursor.

Some JD Edwards EnterpriseOne software applications groups restrict the use of multiple cursors. For example, use multiple cursors only if you have a need for them. Additionally, do not use two cursors to point to the same record at the same time unless both cursors are fetching the record.

Using JDECACHE Partial Keys

A JDECACHE partial key is a subset of a JDECACHE key that is ordered in the same way as the defined index, beginning with the first key in the defined index. For example, for a defined index of N keys, the partial key is the subset of the keys 1, 2, 3, 4...N-1 in that specific order. The order is critical. Partial key components must appear in the same order as the key components in the index. (The index is passed to **jdeCacheInit** or **jdeCacheInitEx**.)

For example, suppose that an index is defined as a structure containing the fields in this order: A, B, C, D, E. The partial keys that can be synthesized from this index are this, in order: A, AB, ABC, ABCD. The previous set is the only set of partial keys that can be synthesized for the defined index: A, B, C, D, E.

A JDECACHE partial key implements the JDECACHE cursor. When you implement the JDECACHE partial key, consider that the JDECACHE cursor works within a JDECACHE data set, which comprises the records within the cache *ordered by* the defined index, *the full index*. If you call a jdeCacheFetchPosition API and pass the partial key, the JDECACHE cursor activates and points to the first record in the JDECACHE data set that matches the partial key. If a **jdeCacheFetchPosition** API was called, subsequent calls to **jdeCacheFetch** will fetch all of the records in the data set that succeed the fetched record *to the end of the data set.* The cursor does *not* stop on the last record that matches the partial key, but continues on to fetch the next record using the next call to **jdeCacheFetch**, even if it does not match the partial key. When a partial key is sent to **jdeCacheFetchPosition**, it merely indicates from where the JDECACHE begins fetching. Because the records in the JDECACHE data set are always ordered, the fetch always retrieves all of the records that satisfy the partial key first.

JDECACHE knows that you are passing a partial key because the fourth parameter to **jdeCacheFetchPosition** indicates the number of key fields that are in the key being sent to the API. If the number of key fields is less than the keys that were indicated when **jdeCacheInit** or **jdeCacheInitEx** was called, then it is a partial key. Suppose the number of keys is N so that JDECACHE uses the first N key fields to make comparisons in order to achieve the partial key functionality. If **jdeCacheFetchPosition** is called with a number of keys that is greater than the number specified on the call to **jdeCacheInit** or **jdeCacheInitEx**, an error is returned.

To delete a partial key, you must make a call to **jdeCacheDeleteAll**. This call deletes all of the records that match the partial key. To indicate to JDECACHE the partial keys that you are using, pass the number of key fields to this API.

Verify that the actual number of key fields in the structure corresponds to the numeric value that describes the number of keys that must be sent to either **jdeCacheFetchPosition** or **jdeCacheDeleteAll**.

CHAPTER 3

Using Business Functions

This chapter provides an overview of business functions (BSFNs) and transaction master business functions, and discusses how to:

- Create transaction master business functions.
- Implement transaction master business functions.
- Work with master file business functions.
- Work with business functions.
- Work with Business Function Builder.
- Work with business function documentation.

Understanding Business Functions

You can use business functions to enhance JD Edwards EnterpriseOne applications by grouping related business logic. Journal Entry Transactions, Calculating Depreciation, and Sales Order Transactions are examples of business functions.

You can create business functions using one of these methods:

- Event rules scripting language.

  The business functions that you create using the event rules scripting language are referred to as Business Function Event Rules (also called Named Event Rules (NERs)). If possible, use NERs for the business functions. In some instances, C business functions might better suit your needs.
- C programming code.

  JD Edwards EnterpriseOne software creates a shell into which you insert logic using C. You use C business functions mainly for caching, but they can also be used for these objects:

  - Batch error level messaging.
  - Large functions.

    C business functions work better for large functions (as determined by the group). If you have a large function, you can break the code up into smaller individual functions and call them from the larger function.
  - Functions for which performance is critical.
  - Complex select statements.

After you create business functions, you can attach them to JD Edwards EnterpriseOne applications to provide additional power, flexibility, and control. You can attach tables and functions to a business function. You must add related tables and functions to the business function object to generate the code for the source and header files. Because the source code for NERs is generated into C, you use the same procedures for debugging both C and NERs.

This section discusses:

- The components of a business function.
- How distributed business functions work.
- C business functions.
- Business function event rules.

Components of a Business Function

The process of creating a business function produces several components. The Object Management Workbench (OMW) is the entry point for the tools that create the components. These components are created:

Component	Where Created
Business Function Specifications	OMW Business Function Design
Data Structure Specifications	OMW Data Structure Design Tool
.C file	Generated in Business Function Design Modified with the IDE
.H file	Generated in Business Function Design Modified with the IDE

The DLLs are divided into categories. This distribution provides better separation between the major functional groups, such as tools, financials, manufacturing, distribution, and so on. Most business functions are organized into a consolidated DLL based on their system code. For example, a financials business function with system code 01 belongs in CFIN.DLL.

Follow these guidelines when you add or modify business functions:

- Create a custom parent DLL unless you are adding a JD Edwards EnterpriseOne business function.

  Assign a parent DLL to the business functions based on the system code defined in UDC table H92/PL. If no DLL is assigned for the system code in which the business function is created, use CCUSTOM, where CUSTOM is the 7-character version of the company name. You can change the DLL after the business function is created.

- When you write business function code, ensure that all calls to other business functions use the jdeCallObject protocol.

  Linker errors might occur if you do not use **jdeCallObject** and you attempt to call a business function in a different DLL. A linker error prevents the function call from working.

Note. If you change the DLL for a business function, go to C:\B9\System\Bin32\BusBuild.exe, select the old DLL file where the business function was, and select Build from the Build menu to rebuild the file.

This table lists some of the DLLs for which Business Function Builder manages the builds:

DLL Name	Functional Group
CAEC	Architecture
CALLBSFN	Consolidate BSFN Library
CBUSPART	Business Partner
CCONVERT	Conversion Business Functions
CCORE	Core Business Functions
CCRIN	Cross Industry Application
CDBASE	Tools - Database
CDDICT	Tools - Data Dictionary
CDESIGN	Design Business Functions
CDIST	Distribution
CFIN	Financials
CHRM	Human Resources
CINSTALL	Tools Install
CINV	Inventory
CLOC	Localization
CLOG	Logistics Functions
CMFG	Manufacturing
CMFG1	Manufacturing - Modification BFs
CMFGBASE	Manufacturing Base Functions
COBJLIB	Tools - Object Librarian
COBLIB	Busbuild Functions
COPBASE	Distribution/Logistic Base Functions
CRES	Resource Scheduling

DLL Name	Functional Group
CRUNTIME	Tools - Run Time
CSALES	Sales Order
CTOOL	Tools - Design Tools
CTRAN	Transportation
CTRANS	Tools - Translations
CWARE	Warehouse
CWRKFLOW	Tools - Workflow
JDBTRG1	Table Trigger Library 1
JDBTRG2	Table Trigger Library 2
JDBTRG3	Table Trigger Library 3
JDBTRG4	Table Trigger Library 4
JDBTRIG	Parent DLL for Database Triggers

Note. Do not use table triggers for regular business functions.

How Distributed Business Functions Work

OMW manages these three main components that make up NERs or business functions:

- Object Name

  The Object Name is the actual source file.
- Function Name

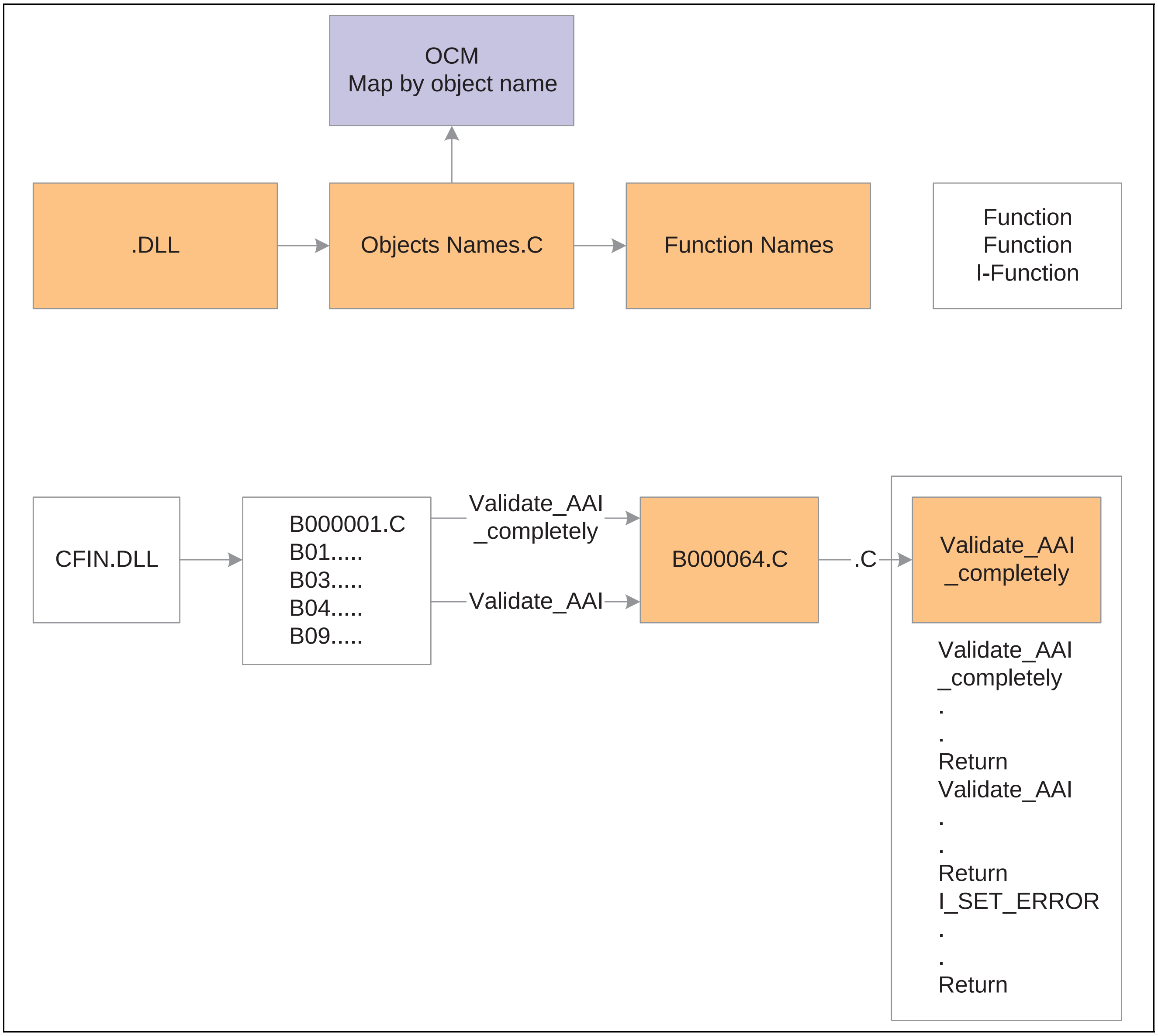
  The name of the business function or event rule.

  Note. Any business function, whether it uses C or NERs as its source language, must have a defined data structure to send or receive parameters to or from applications. You can create a DSTR data structure object, or select an existing object type to work with in OMW. You can also create data structures for text substitution messages. Additionally, you can attach notes, such as an explanation of use, to any data structure or data item within the structure.
- DLL Name

  The DLL is a dynamic link library.

When a business function is called, the Object Configuration Manager (OCM) determines where to run the business function. After the system maps a business function to a server, calls from that business function cannot be mapped back to the workstation.

This flowchart illustrates how distributed business functions work:



Distributed business function

C Business Functions

JD Edwards EnterpriseOne software contains two types of business functions: NERs and C business functions. C business functions are written in C programming language and are used to perform functions that are not available in NERs. C business functions include both a header file (.h) and a source file (.c).

Header File Sections

This table describes the major sections of a business function header file:

Section	What It Includes	Description and Guidelines
Header File Comment	• Header file name • Description • History • Programmer • SAR number • Copyright information	Comments that the input process of the Business Function Source Librarian builds. The programmer name and SAR number are manually updated by the programmer.
Table Header Inclusions	Include statements for header files associated with tables that are directly accessed by this business function.	Table header files include definitions for the fields in a table and the ID of the table itself.
External Business Function Header Inclusions	Include statements for headers associated with externally defined business functions that are directly accessed by this business function.	External function calls with **jdeCallObject** are included to use the predefined data structures.
Global Definitions	Global constants used by the business function.	Use global definitions sparingly. They include symbolic names that you enter in uppercase; words are separated by an underscore character.
Structure Type Definitions	Data structure definitions for internal processing.	To prevent naming conflicts, define this structure using structure names that are prefixed by the source file name.
DS Template Type Definition	Data structure type definitions generated by Business Function Design. Symbolic constants for the data structure generated by Business Function Design.	Modify this structure through OMW.
Source Preprocessor	• Undefines JDEBFRTN if it is already defined. • Checks for how to define JDEBFRTN. • Defines JDEBFRTN.	Ensures that the business function declaration and prototype are properly defined for the environment and source file, including this header.
Business Function Prototype	Prototypes for all business functions in the source file.	Defines the business functions in the source file, the parameters that are passed to them, and the type of value that they return.
Internal Function Prototype	Prototypes for all internal functions that are required to support business functions within this source file.	Defines the internal functions that are associated with the business functions in the source file, the parameters that are passed to each internal function, and the type of value that they return.

Example: Business Function Header File

Assume that Business Function Design created this header file. This file contains only the required components in a business function header file:

```
Header File Begin
/*****************************************************************************
* Header File: B99TEST.h
*
* Description: test Header File
*
* History:
*  Date Programmer SAR# - Description
*  ---------- ---------- ------------------------------------------
* Author 10/14/2003 DEMO Unknown - Created
*
*
* Copyright (c) 1994 Oracle 2003
*
* This unpublished material is proprietary to Oracle.
* 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 Oracle.
****************************************************************************/
#ifndef __B99TEST_H
#define __B99TEST_H
/*****************************************************************************
* Table Header Inclusions
****************************************************************************/
/*****************************************************************************
* External Business Function Header Inclusions
****************************************************************************/
/*****************************************************************************
* Global Definitions
****************************************************************************/
/*****************************************************************************
* Structure Definitions
****************************************************************************/
/*****************************************************************************
* DS Template Type Definitions
****************************************************************************/
/*****************************************
* TYPEDEF for Data Structure
* Template Name: Test Data Structure
* Template ID: D59TEST
* Generated: Tue Oct 14 16:53:08 2003
*
* DO NOT EDIT THE FOLLOWING TYPEDEF
* To make modifications, use the EnterpriseOne Data Structure
* Tool to Generate a revised version, and paste from
```



```
* the clipboard.
*
**************************************/
#ifndef DATASTRUCTURE_D59TEST
#define DATASTRUCTURE_D59TEST
typedef struct tagDSD59TEST
{
 JCHAR  cEverestEventPoint01;
 JCHAR  szNameAlpha[41];
 MATH_NUMERIC mnAmountField;
} DSD59TEST, *LPDSD59TEST;
#define IDERRcEverestEventPoint01_1  1L
#define IDERRszNameAlpha_2   2L
#define IDERRmnAmountField_3   3L
#endif
/*****************************************************************************
* 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 F0101Test
 (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSD0100018 lpDS);
/*****************************************************************************
* Internal Function Prototypes
****************************************************************************/
#endif /* __B99TEST_H */
Header File End
```

This table describes the contents of the various lines in the header file:

Header File Line	Where Input	Description
Header File	OMW	Verify the name of the business function header file.
Description	OMW	Verify the description.

Header File Line	Where Input	Description
History	IDE	Manually update the modification log with the programmer name and the appropriate SAR number.
#ifndef	Business Function Design	Symbolic constant prevents the contents from being included multiple times.
Table Header Inclusion	Business Function Design	When business functions access tables, related tables are input and Business Function Design generates an include statement for the table header file.
External Business Function Header Inclusions	Business Function Design	No external business functions for this application.
Global Definitions	IDE	Constants and definitions for the business function. It is not recommended that you use this block. Global variables are not recommended. Global definitions go in .c not .h.
Structure Definitions	IDE	Data structures for passing information between business functions, internal functions, and database APIs.
TYPEDEF for Data Structure	Business Function Design	Data structure type definition. Used to pass information between an application or report and a business function. The programmer places it on the clipboard and pastes it in the header file. Its components include: • Comment Block, which describes the data structure. • Preprocessor Directives, which ensure that the data type is defined only once. • Typedef, which defines the new data type. • #define, which contains the ID to be used in processing if the related data structure element is in error. • #endif, which ends the definition of the data structure type definition and its related information.

Header File Line	Where Input	Description
Source Preprocessor Definitions	Business Function Design	All business function header files contain this section to ensure that the business function is prototyped and declared based on where this header is included.
Business Function Prototype	Business Function Design	Used for prototypes of the business function.
JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode	Business Function Design	Business Function Standard All business functions share the same return type and parameter data types. Only the function name and the data structure number vary between business functions. Parameters include: • *LPBHVRCOM* Pointer to a data structure used for communicating with business functions. Values include an environment handle. • *LPVOID* Pointer to a void data structure. Currently used for error processing; will be used for security in the future. • *LPDS#####* Pointer to a data structure containing information that is passed between the business function and the application or report that invoked it. This number is generated through Object Librarian. • *JDEBFRTN(ID)JDEBFWINAPI* All business functions will be declared with this return type. It ensures that they are exported and imported properly. Parameter names (*lpBhvrCom*, *lpVoid*, and *lpDS*) will be the same for all business functions.
Internal Function Prototypes	Business Function Design	Internal function prototypes required to support the business functions in this source file.

Source File Sections

OMW builds a template for the business function source file. The business function source file consists of several major sections, as described in this table:

Section	What It Includes	Description
Source File Comment Block	• Source file name • Description • History • Programmer • Date • SAR Number • Description • Copyright information	Built from the information in the Business Function Design Tool. The programmer manually updates the programmer name and SAR number.
Notes Comment Block	Any additional relevant notes concerning the business function source.	Document complex algorithms used, how the business functions in the source relate to each other, and so on.
Business Function Comment Block	• Business function name • Description • Description list of the parameters	
Business Function Source Code	Source code for the business function.	
Internal Function Comment Block	• Function name • Notes • Returns • Parameters	Copy these blocks and place the values in the specified sections to describe the internal function. Follow the comment block with internal function source code.
Internal Function Source Code	Source code for the internal function described in the comment block.	The business function developer enters this code as needed. A populated internal function comment block must precede this code.

Example: Business Function Source File

Assume that Business Function Design created this source file called Check for In Add Mode. It contains the minimum components required in a business function source file. The source code in the Main Processing section is entered manually, and varies from business function to business function. All other components are generated by Business Function Design.

```
#include <jde.h>

#define b98sa001_c

/*****************************************************************************
 * Source File: B98SA001.c
```



```
 *
 * Description: Check for In Add Mode Source File
 ****************************************************************************/
 **************************************************************************/

#include <b98sa001.h>

/**************************************************************************
 * Business Function: CheckForInAddMode
 *
 *  Description: Check for In Add Mode
 *
 *   Parameters:
 *   LPBHVRCOM   lpBhvrCom Business Function Communications
 *   LPVOID    lpVoid  Void Parameter - DO NOT USE!
 *   LPDSD98SA0011   lpDS   Parameter Data Structure Pointer
 *
 *************************************************************************/

JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid,⇒
 LPDSD98SA0011 lpDS)
{
 /************************************************************************
 * Variable declarations
 ************************************************************************/

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

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

 /************************************************************************
 * Check for NULL pointers
 ************************************************************************/
 if ((lpBhvrCom == NULL) ||
  (lpVoid == NULL) ||
  (lpDS  == NULL))
 {
  jdeSetGBRError (lpBhvrCom, lpVoid, (ID) 0, _J("4363"));
  return CONTINUE_GBR;
 }

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

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



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

 if (lpBhvrCom->iBobMode == BOB_MODE_ADD)
 {
   lpDS->cEverestEventPoint01 = _J('1');
 }
 else

 {
   lpDS->cEverestEventPoint01 = _J('0');
 }

 return (BHVR_SUCCESS);
}

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

The lines that appear in the source file are described in this table:

Source File Line	Where Input	Description and Guidelines
#include <jde.h>	Business Function Design	Includes all base JD Edwards EnterpriseOne definitions.
#define b98sa001_c	Business Function Design	Ensures that related header file definitions are correctly created for this source file.
Source File	OMW	Verifies the information in the file comment section. Enter the programmer's name, SAR number, and description.
#include <B98SA001.h>	OMW	Includes the header file for this application.
Business Function	Business Function Design	Verifies the name and description in the business function comment block.

Source File Line	Where Input	Description and Guidelines
JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDS104438 lpDS)	Business Function Design	Includes the header of a business function declaration.
Variable declarations	IDE	Declares variables that are local to the business function.
Declare structures	IDE	Declares local data structures to communicate between business functions, internal functions, and the database.
Declare pointers	IDE	Declares pointers.
Check for NULL pointers	Business Function Design	Business Function Standard Verifies that all communication structures between an application and the business function are valid.
jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, _J("4363"), LPVOID) NULL); return ER_ERROR;	Business Function Design	Sets the standard error to be returned to the calling application when any of the communication data structures are invalid.
Set pointers	IDE	Declares and assigns appropriate values to pointers.
Main Processing	IDE	Provides main functionality for a business function.
Function Clean Up	IDE	Frees any dynamically allocated memory.
Internal function comment block	IDE	Defines internal functions that are required to support the business function. They should follow the same C coding standards. A comment block is required for each internal function and should be formatted correctly.

Use the MATH_NUMERIC data type exclusively to represent all numeric values in JD Edwards EnterpriseOne software. The values of all numeric fields on a form or batch process are communicated to business functions in the form of pointers to MATH_NUMERIC data structures. MATH_NUMERIC is used as a data dictionary (DD) data type.

Business Function Event Rules

A NER is a business function object for which the source language is event rules instead of C. You create a NER using the event rules scripting language. This scripting language is platform-independent and is stored in a database as a JD Edwards EnterpriseOne software object. NERs are modular. That is, they can be reused in multiple places by multiple programs. This modularity reduces rework and enables you to reuse code.

Not all chunks of code should be packaged in a business function module. For example, when code is so specific that it applies only to a particular program, and it is not reused by any other programs, you should leave it in one place instead of packaging it in a business function. You can attach all the logic on a hidden control (**Button Clicked** event) and use a system function to process the logic as needed.

An example of a NER is N3201030. This business function creates generic text and Work Order detail records (for the F4802 table) for a configured work order. Based on the structure of the sales order in the F3296 table, the configured segments for the item on the passed work order and all lower level segments are included in the generic text.

This example illustrates the function as it appears in Event Rules Design:

```
Named Event Rule Begin
//
// Convert the related sales order number into a math numeric. If that fails
// exit the function
//
String, Convert String to Numeric
If VA evt_cErrorCode is equal to "1"
//
// Validate that the work order item is a configured item.
//
F4102 Get Item Manufacturing Information
If VA evt_cStockingType is not equal to "C"
 And BF cSsuppressErrorMessages is not equal to "1"
BF szErrorMessageID = "3743"
Else
BF szErrorMessageID = " "
//
// Delete all existing "A" records from F4802 for this work order.
//
VA evt_cWODetailRecordType = "A"
F4802.Delete
F4802.Close
//
// Get the segment delimiter from configurator constants.
//
F3293 Get Configurator Constant Row
If VA evt_cSegmentDelimiter is less than or equal to <Blank>
VA evt_cSegmentDelimiter - /
End If
//
F3296.Open
F3296.Select
If SV File_IO_Status     is equal to CO SUCCESS
```

```
F3296.FetchNext
//
// Retrieve the F3296 record of the work order item. and determine its key
// sequence by parsing ATSQ looking for the last occurrence of "1". The substring
// of ATSQ to this point becomes the key for finding the lower level configured
// strings
//
If VA evt_mnCurrentSOLine is equal to BF mnRelatedSalesOrderLineNumber
// Get the corresponding record from F32943. Process the results of that fetch
// through B3200600 to add the parent work order configuration to the work order
// generic text.
F32943.FetchSingle
If SV File_IO_Status   is equal to CO SUCCESS
VA evt_szConfiguredString = concat([VA evt_ConfiguredStringSegment01],
[VA evt_ConfiguredStringSegment02])
Confg String Format Segments Cache
End If
//
// Find the last level in ATSQ that is not "00". Note that the first three
// characters represent the SO Line Number to the left of the decimal.
Example:
// SO Line 13.001 will have the ATSQ characters "013". Each configured item can⇒
 have
// 99 lower-level P-Rule items and a total of ten levels. Therefore every pair
// thereafter is tested.
//
VA evt_mnSequencePosition - 1
While VA evt_mnSequencePosition is less than "23"
And VA evt_szCharacterPair is not equal to "00"
VA evt_mnSequencePosition - [VA evt_mnSequencePosition] + 2
VA evt_szCharacterPair = substr([VA evt_szTempATSQ],[VA evt_mnSequencePostion],2)
End While
VA evt_szParentATSQ = substr([VA evt_szTempATSQ],0,[VA evt_mnSequencePosition])
//
// For each record in F3296 for the related sales order, find those with the same
// key substring of ATSQ. Retrieve the associated record from F32943 if
// available and pass the configured string to N3200600 for addition to the work
// order generic text.
//
F3296.FetchNext
Wile SV File_IO_Status  is equal to CO SUCCESS
VA evt_szChildATSQ = substr([VA evt_szTempATSQ],0,[VA evt_mnSequencePosition]}
If VA evt_szChildATSQ is equal to VA evt_szParentATSQ
F32943.FetchSingle
If SV File_IO_Status   is equal to CO SUCCESS
VA evt_szCongifuredString = concat([VA evt_ConfiguredStringSegment01],
[VA evt_ConfiguredStringSegment02])
Confg String Format Segments Cache
End If
End If
```



```
F3296.FetchNext
End Whil
F32943.Close
//
// Unload segments cache into the work order generic text. B3200600 Mode 6
Confg String Format Segments Cache
//
End If
End If
F3296.Close
//
End If
Else
// The related sales order number is invalid. Return an error.
If BF cSuppressErrorMessages is not equal to "1"
Set NER Error ("0002", BF SzRelatedSalesOrderNumber)
End If
End Ir
Named Event Rule End
```

Understanding Transaction Master Business Functions

Transaction master business functions provide a common set of functions that contain all of the necessary default values and editing for a transaction table in which records depend on each other. Transaction master business functions contain logic that ensures the integrity of the transaction being inserted, updated, or deleted from the database. Event flow breaks up logic. You use cache APIs to store records that are being processed. You should consider using a transaction master business function in these situations:

- You accept transaction file records from a non-JD Edwards EnterpriseOne source.
- Multiple applications update the same transaction file.

These transaction tables are examples of candidates for transaction master business functions:

- The F0911 table accepts updates across application suites, as well as external sources.
- The F06116 table accepts updates from batch, interactive, and external sources.

A master business function (MBF) can be called from several different applications. Rather than duplicating the processing options for the MBF on each application, you typically create a separate processing option template for these processing options. You can use interactive versions to set up different versions of the MBF processing options. Various calling programs then pass the version name to the version parameter of **BeginDoc**.

From within **BeginDoc**, the business function **AllocatePOVersionData** can be called to retrieve the processing options by version name. The processing options needed by other modules can be written to the header cache and accessed later, rather than calling **AllocatePOVersionData** multiple times.

The cache structure stores all lines of the transaction. Transaction lines are written to the cache after they have been edited. The **EndDoc** module then reads the cache to update the database.

This table describes the components of the header section:

Field Description	Field	Key	Type	Size
Job Number	JOBS	X	Num	
Document Action	ACTN		Char	1
Processing Options				
Currency Flag	CYCR		Char	1
Business View Fields				
Work Fields				

This table explains the fields:

Field Description	Purpose
Job Number	A unique system-assigned number assigned when the BeginDoc module starts the job. This distinguishes transactions in the cache for each job on the workstation that is using the cache. Use next number 00/4 for the job number. If you are using a unique cache name (D*xxxxxxxxx[job number]*), you do not necessarily need the job number field stored in the cache for a key because you would only be working with one transaction per cache. You can, therefore, use any field as the key to the cache.
Document Action	The action for the document. Values are: • A or 1 = Add • C or 2 = Change • D = Delete
Processing Options	Processing option values were read in using **AllocatePOVersionData**, and are needed in other modules of the MBF.
Currency Flag	A system value that indicates whether currency is on and what method of currency conversion is used (N, Y, or Z).
Business View Fields	The fields required for processing the transaction and writing it to the database. All fields in the record format that are not saved in the header cache will be initialized when the record is added to the database using the APIs.
Work Fields	Fields that are not part of the business view (BV), but are needed for editing and updating the transaction. For example, Last Line Number is the last line number written to the detail cache. It will be stored at the header level, and retrieved and incremented by the MBF. The incremented line number will be passed to the header cache and stored for the next transaction.

This table describes the components of the detail section:

Field Description	Field	Key	Type	Size
Job Number	JOBS	X	Char	8
Line Number	(Application-specific)	X	Num	
Line Action	ACTN		Char	1
Business View Fields				
Work Fields				

This table explains the fields:

Field Description	Purpose
Job Number	A unique number assigned when the **BeginDoc** module starts the job. This distinguishes transactions in the cache for each job on the client that is using the cache. If you are using a unique cache name (D*xxxxxxxx[job number]*), you do not necessarily need to store the job number field in the cache for a key because you work with only one transaction per cache. You can, therefore, use line number only as the key to the cache.
Line Number	The number used to uniquely identify lines in the detail cache. This line number can also eventually be assigned to the transaction when it is written to the database. The transaction lines are written to the detail cache only if they are error-free.
Line Action	The action for the transaction line. Values are: • A or 1 = Add • C or 2 = Change • D = Delete
Business View Fields	Fields required for processing the transaction that will be written to the database. All fields in the record format that are not saved in the detail cache will be initialized when the record is added to database using the APIs.
Work Fields	Fields that are not part of the business view, but are needed for editing and updating the transaction line.

Building Transaction Master Business Functions

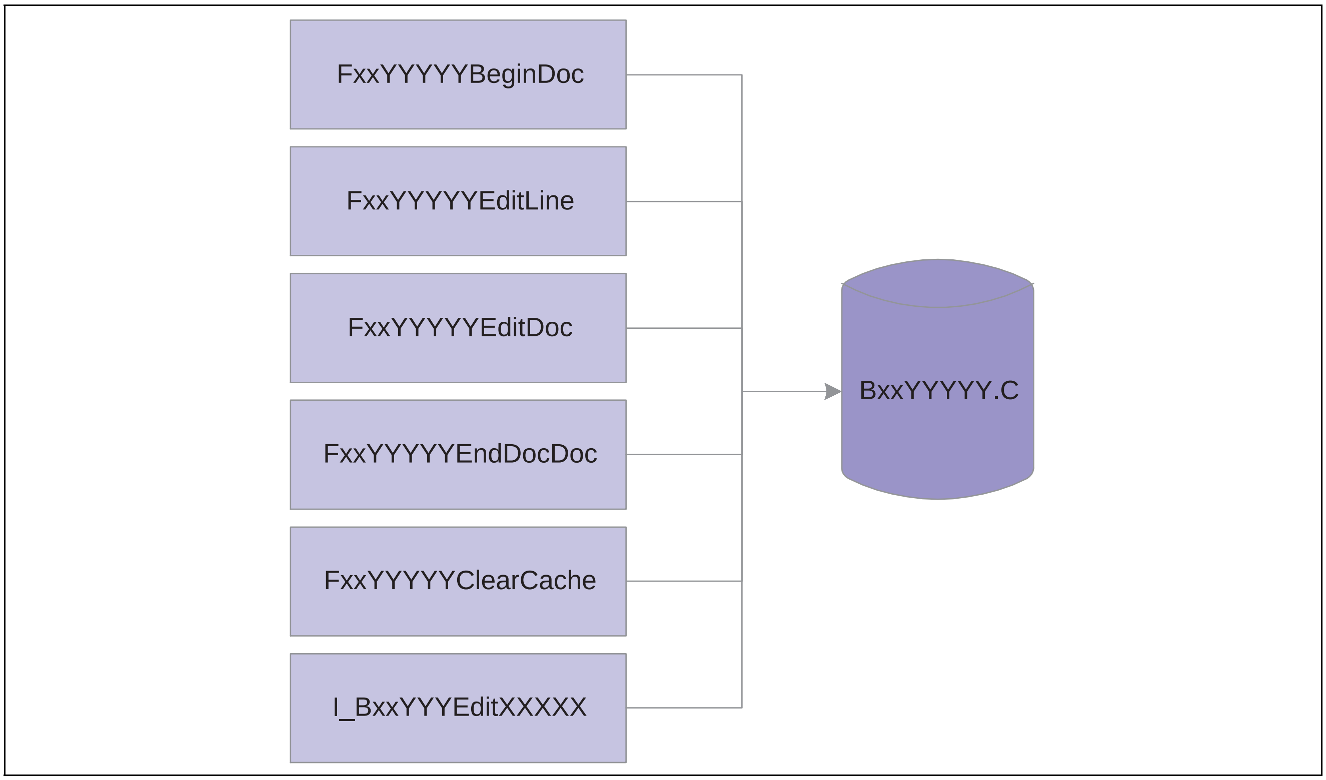
This section provides an overview of building transaction master business functions, and discusses the component used to build such a business function:

- Begin document
- Edit line
- Edit document
- End document
- Clear cache
- Cancel document

Understanding Building Transaction Master Business Functions

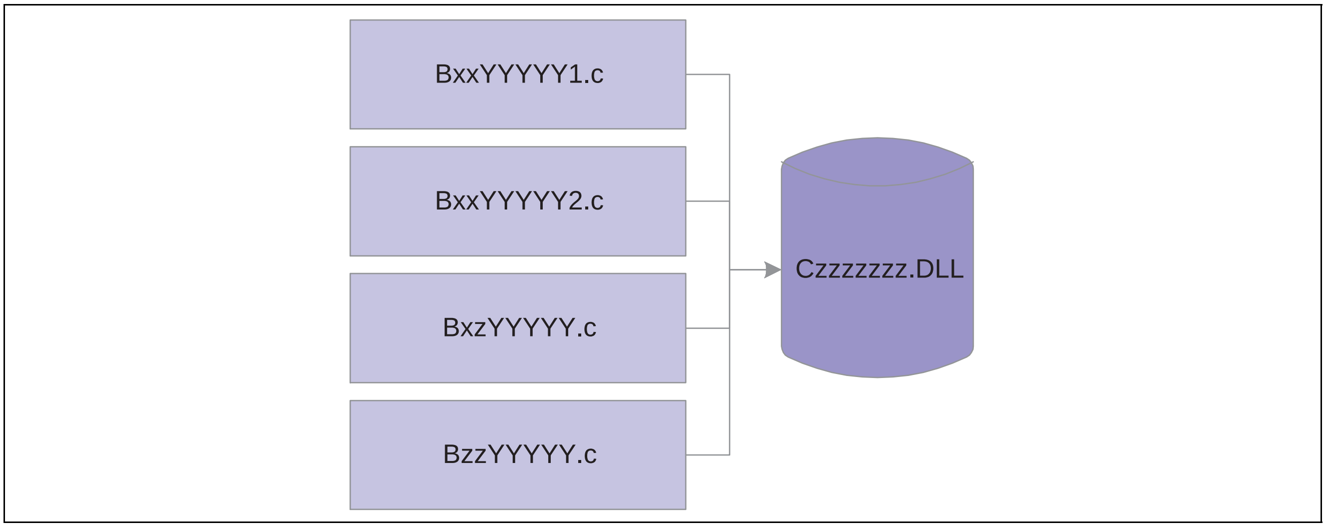
These flowcharts illustrate how transaction master business functions are built.

First, you create the individual business functions using several basic components:



Building transaction master business functions

Next, you combine the business functions into a DLL:



Combining business functions into a .DLL

You typically use these basic components to create a master business function as described by this table:

Component	Purpose
Begin Document	Called when all header information has been entered. Creates initial header if it has not already been created. Can also include default values, editing, and processing options (POs).
Edit Line	Called when all line information has been entered. Creates cache for detail information if it has not already been created.
Edit Document	Called when ready to commit the transaction. Processes any remaining document edits and verifies that all records are valid to commit.
End Document	Called when you need to commit the transaction. Processes all records in the header and detail cache, performs I/O, and deletes caches.
Clear Cache	Called when you are ready to delete all cache records. Deletes header and detail cache records.

Begin Document

Begin Document has this format:

```
FxxxxxBeginDocument
```

The Begin Document component performs these tasks:

- Inserts default information and edits information in the header, including data dictionary defaults and UDC editing.
- Fetches information from the database, if necessary, to ensure that the selected document action can take place.
- Validates and processes information that is common to all records.
- Writes the record to header cache if no errors exist.

- Contains all header cache information that is common to all detail records. This improves performance by eliminating the need to use all the detail records to perform the same validations and table I/O.
- Updates the header cache with the new information when information in the header fields changes and Begin Document has previously been called.

Special Logic or Processing Required

On the initial call, the function assigns the job number. To retrieve the job number, this function calls X0010GetNextNumber with a system code of 00 and an index number of 04. If called again, Begin Document passes the job number that was previously assigned; therefore, it does not need to assign another job number.

Hook Up Tips

Keep these tips in mind when calling Begin Document:

- You must call a function at least once before calling Edit Line.
- If errors occur during validation of the header field when the function is called, call the function again to verify that errors have been cleared before calling Edit Line.
- If this function might be called multiple times from different events, include it on a hidden button on an application to reduce duplicate code and ensure consistency. This button might then be called from focus on grid because the user is then adding or deleting detail records, and is finished adding header information. In case of a Copy in which the user does not use the grid, this button might also be called on OK button.
- Calling a button from an asynchronous event breaks the asynchronous flow and forces the button to be processed in synchronous mode (inline).

Common Parameters

This table describes the common parameters for Begin Document:

Name	Alias	I/O	Description
Job Number	JOBS	I/O	Pass Job Number created in Begin Document, if previously called; otherwise, pass zeros and assign a job number.
Document Action	ACTN	I	A or 1 = Add C or 2 = Change D = Delete This is the action of the entire Document, not the individual detail lines. For example, you might modify a few detail lines in Edit Line, add a few detail lines in Edit Line, and delete a few detail lines in Edit Line, but the Document Action in Begin Document would be Change.

Name	Alias	I/O	Description
Process Edits	EV01	I	Optional 0 = No Edits Any Other = Full Edits **Note.** The GUI interface usually uses the partial edit, and the batch interface uses the full edit. If you leave this parameter blank, the default option is full edits.
ErrorConditions	EV02	O	Blank = No Errors 1 = Warning 2 = Error
Version	VERS	I	This field is required if this MBF is using versions.
Header Field One	****	I/O	Pass in all the header fields that are common to the entire document. Begin Document processes all of these fields and validates them, data dictionary edits, UDC editing, default values, and so on. Begin document might also fetch to the table to validate that records matching these header fields exist for Delete and Change, or do not exist for Add.
Header Field Two	****	I/O	
. . .	****	I/O	
Header Field XX	****	I/O	
Work Field / Processing Flag One	****	I	List any work fields that the program needs. These could be flags for processing, dates to validate, and so on. These fields might or might not be used. For example, currency control might be saved in the header cache so that all detail records would either use currency or not.

Name	Alias	I/O	Description
Work Field / Processing Flag One	****	I	
. . .		I	
Work Field / Processing Flag One		I	

Application-Specific Parameters

Application-specific parameters must perform these tasks:

- List the fields that are needed to process header-level information.
- List any work fields that are needed to perform edits.
- List all POs that are needed to process header-level information.

Edit Line

Edit Line has this format:

```
FxxxxxEditLine
```

The Edit Line component performs these tasks:

- Validates all user input, performs calculations, and retrieves default information.

  Edit Line is normally called for every record that is fetched. It performs the edits for that *one* record in the file.
- Reads header cache records for default values.
- On an ADD, enters default information in blank columns, such as address book information.

  The default values might come from any of these objects:
  - Another column in the line.
  - A process performed on a column sent in the line.
  - A PO.
  - A saved value from the header record that was determined in the Begin Document module.
  - A DD default value.
- Edits columns for correct information.

  This includes interdependent editing between columns. Also performs UDC and DD edits.
- Writes record to the detail cache if no errors occurred.

  If the record already exists in the work file, the line in the work file will be retrieved and updated with the changes. If a record is deleted from the grid in direct mode, and the record does not exist in the database, the record will be removed from the detail cache. If the record exists in the database, the action code for the record will be changed to delete, and the record will be stored in the detail cache until file processing in End Doc.

Special Logic or Processing Required

Depending on the type of document being processed, different editing and inserting of default values takes place. An example would be vouchers and invoices processed through the journal entry MBF. The tax calculator is only called for vouchers. Depending on the event processing required, the process edit flag determines the editing that occurs. For example, in an interactive program, when the *Grid Record is Fetched* event runs, Partial Edits might be performed to retrieve descriptions, default values, and so on. When the *Row is Exited and Changed* event runs, Full Edits might be performed to validate all user input.

Typical Uses and Hookup

In interactive applications, Edit Line is typically called on Grid Record is Fetched or Row is Exited and Change (Asynch). In batch applications, Edit Line is typically called in the Do section of the group, columnar, or tabular section.

Common Parameters

This table describes the common parameters for Edit Line:

Name	Alias	I/O	Description
Job Number	JOBS	I	Used as key or to create a unique name for the cache or work file. Retrieved from Begin Document.
Line Number	LNID	I/O	The unique number identifying the transaction line. Can also be used as the line number in the Detail Cache.
Line Action	ACTN	I	A or 1 = Add C or 2 = Change D or 3 = Delete
Process Edits (optional)	EV01	I	0 = No Edits 1 = Full Edits 2 = Partial Edits **Note.** GUI interface typically uses the partial edit, and the batch interface typically uses the full edit. If you leave this parameter blank, the default edit is Full.
Error Conditions	ERRC	O	0 = No Errors 1 = Warning 2 = Error

Name	Alias	I/O	Description
Update Or Write to Work File	EV02	I	1 = Write or update records to the work file, or do both.
Record Written to Work File	EV03	I/O	1 = A record is written to the work file. This reduces I/O calls to the work file. Blank = No record is written to the work file.
Detail Field One	****	I/O	Pass in all the Detail fields that will be edited. Typically, these are the grid record fields. Edit Line provides validation, data dictionary edits, UDC editing, default values, and so on.
Detail Field Two	****	I/O	
Detail Field XX	****	I/O	
Work Field / Processing Flag One	****	I	List any work fields that the program needs. These fields could be flags for processing, dates to validate, and so on.
Work Field / Processing Flag One	****	I	
Work Field / Processing Flag One	****	I	

Edit Document

The Edit Document component performs these tasks:

- Reads cache records if multiple line editing is required.
- Reads header cache record if header information is needed.
- Performs cross-dependency edits involving multiple lines in a document. For example, Edit Document processes all records to ensure that percentages total 100 percent, and it ensures that the last record does not contain certain information.

Special Logic or Processing Required

Depending on the type of document that you are processing, different logic is executed. For example, vouchers and invoices are processed through the journal entry edit object, although the balancing is different for these document types.

Hook Up Tips

Edit Document is typically used in this fashion:

- Call the function at least once after calling Edit Line and before End Document.
- If errors occur during validation, call the function again to verify that errors have been cleared before calling End Document.
- Call this function on the OK *Button Clicked* event so that, if errors do occur, they are corrected before the user exits the application.

Common Parameters

This table describes the common parameters for Edit Document:

Name	Alias	I/O	Description
Job Number	JOBS	I	Retrieved from Begin Document
ErrorConditions	EV01	O	Blank = No Errors 1 = Warning 2 = Error

Application-Specific Parameters

Because all records have been added in Begin Document or Edit Line, and because any information needed to process the entire document is in cache, few parameters are needed in this function.

End Document

End Document has this format:

```
FxxxxxEndDocument
```

The End Document component performs these tasks:

- Assigns a next number to the document.

  For vouchers, you should do this before calling journal entry edit object, but not before the voucher has been balanced and is ready to be added to the database. By placing this module on the before add/delete/update events, the document passes all edits before running this event.
- Reads cache records.
- On an ADD, writes new rows to the table.
- On a CHG, retrieves and updates existing rows.
- On a DEL, deletes rows from the table.
- Adds information and updates associated tables.

  For example, it adds and updates these objects:
  - Manual checks associated with vouchers.
  - Address Book vouchered YTD columns in Address Book.

  - Address, phones, and who's who information for Address Book.
  - Batch header.
- Clears the cache for that document and any work fields after all updates are completed successfully.
- Summarizes documents, if designated in a processing option, as it writes to the database.
- Reads work file through an alternate means and writes the records at a control break.
- Performs currency conversion.

Hook-Up Tips

This function is typically called on OK button **Post Button Clicked**, and it is hooked up Asynch. In the C code, after the insert or update to the database is successful, call **Clear Cache** to clear the cache.

Common Parameters

This table describes the common parameters for End Document:

Name	Alias	I/O	Description
Job Number	JOBS	I	Retrieved from Begin Document
Computer ID	CTID	I	Retrieved from GetAuditInfo(B9800100) in application (optional)
Error Conditions	EV01	O	Blank = No errors 1 = Warning 2 = Error
Program ID	PID	I	Usually hard-coded

Application-Specific Parameters

Use application-specific parameters in End Document to perform these tasks:

- List the fields that are needed to process update or writes, such as Time and Date Stamp fields.
- List any work fields that are needed to perform updates or writes.
- List all POs that are needed to process updates or writes.

Clear Cache

Clear Cache has this format:

```
FxxxxxClearCache
```

The Clear Cache component removes the records from the header and detail cache.

Special Logic or Processing Required

If a unique cache name is selected as the naming convention for the cache (`D`*xxxxxxxx*[*Job Number*]), then use the cache API **jdeCacheTerminateAll** to destroy the cache.

Common Parameters

This table describes the common parameters for Clear Cache:

Name	Alias	I/O	Description
Job Number	JOBS	I	Indicates the job number of the transaction that you want to clear. This job number should have been returned from BeginDoc.
Clear Header	EV01	I	Indicates whether the header cache should be cleared. 1 = clear cache
Clear Detail	EV02	I	Indicates whether the detail cache should be cleared 1 = clear cache
Line Number From (Optional)	LNID	I	Indicates where to begin clearing records in the detail cache. If this line is blank, the system begins clearing from the first record.
Line Number Thru (Optional)	NLIN	I	Indicates where to stop clearing records in the detail cache. If this line is blank, the system deletes to the end of the cache.

Cancel Document

Cancel Document has this format:

```
FxxxxxxCancelDoc
```

The optional Cancel Document component is used primarily with the Cancel button to close files, clear the cache, and so on. Cancel Document is an application-specific function that provides basic function cleanup.

Special Logic or Processing Required

This function is application-specific.

Common Parameter

This table describes the common parameter for Cancel Document:

Name	Alias	I/O	Description
Job Number	JOBS	I	The job number of the transaction that you want to clear. This number should have been returned from BeginDoc.

Implementing Transaction Master Business Functions

This section discusses using single-record processing and document processing to implement transaction master business functions.

Single-Record Processing

This section provides an interactive and a batch program flow example for single-record processing.

Interactive Program Flow Example

This is an example of an implementing transaction master business functions during single-record processing in an interactive application:

1. **Post Dialog is Initialized (optional)**

   Call Begin Document.
2. **Set Focus on Grid**
3. **Row is Exited and Changed** or **Row is Exited and Changed ASYNC**

   Call Edit Line.
4. **Delete Grid Record Verify- After**

   Call Edit Line to perform delete for one record.

   Call Edit Document to perform deletes on a group of records.
5. OK **Button Clicked**

   Call Begin Doc.

   Call Edit Document.
6. OK **Post Button Clicked**

   Call End Document.

Master Business Functions usually perform all table I/O for the given table. Therefore, these actions must be disabled:

- **Add Grid Record to DB - before**

  Suppress Add.
- **Update Grid Record to DB - before**

  Suppress Update.
- **Delete Grid Record to DB - before**

Suppress Delete.

Batch Program Flow Example

This is an example of an implementing transaction master business functions during single-record processing in a batch application:

1. Do Section of Report Header.

   Call Begin Document.

2. Do Section of the Group Section.

   Call Edit Line.

3. Do Section of a Conditional Section (optional).

   Call Edit Document.

4. Do Section of Report Footer.

   Call End Document.

Document Processing

This section provides an interactive program flow example for document processing.

Program Flow Example

This is an example of an implementing transaction master business functions during document processing in an interactive application:

1. **Dialog is Initialized**

   Call Open Batch Edit Object module.

2. **Grid is Entered**

   Call Begin Document Edit Object module.

3. **Row is Exited**

   Call Edit Line Edit Object module.

4. OK **Button Clicked**

   Call Edit Document Edit Object module.

5. **Before Add from Database** or **Before Delete from Database**

   Suppress Add/Delete.

   Call End Document Edit Object module.

6. Cancel **Button Clicked**

   Call Close Batch Edit Object module.

Working with Master File Master Business Functions

Master business functions (MBFs) enable calling programs to process certain predefined transactions. An MBF encapsulates the required logic, enforces data integrity, and insulates the calling programs from the database structures. Use MBFs for these reasons:

- To create reusable, application-specific code.
- To reduce duplicated code.
- To ensure that hookup is consistent.
- To support interoperability models.
- To enable processing to be distributed through OCM.
- To design event-driven architecture.

MBFs are typically used for multiline business transactions such as journal entries or purchase orders. However, certain master files also require MBF support due to their complexity, importance, or maintenance requirements from external parties. The requirements for maintaining master files are different from those for multiline business transactions.

Generally, master file MBFs are much simpler than multiline business transaction MBFs. Transaction MBFs are specific to a program, while master file MBFs access a table multiple times.

For interoperability, master file MBFs can be used instead of table I/O. This enables you to perform updates to related tables using the business function instead of table event rules. Multiple records are not used; instead, all edits and actions are performed with one call.

In their basic form, master file MBFs have these characteristics:

Characteristic	Description
Single call	Generally, you can make one call to an MBF to edit, add, update, or delete a master file record. An edit-only option is available also.
Single data structure	The fields required to make the request and provide all the necessary values are in one data structure. The data fields should correspond directly with columns in the associated master file.
No cache	Because each master file record is independent of the others, caching is unnecessary. The information provided with each call and the current condition of the database provides all of the information that the MBF needs to perform the requested function.
Normal error handling	As with other MBFs, master file MBFs must be capable of executing both in interactive and batch environments. Therefore, the calling program must determine the delivery mechanism of the errors.
Inquiry feature	To enable external systems to be insulated from the JD Edwards EnterpriseOne database, an inquiry option is included. This enables an external system to use the same interface to access descriptive information about a master file key as it uses to maintain it.
Effect on applications	For JD Edwards EnterpriseOne applications, the effect of implementing a master file MBF should be minimal. Consider and follow several standards before implementing a master file MBF.

Master file applications use the system to process all I/O for find/browse forms. This enables you to use all of the search capabilities of the software.

You should design all master file applications so that all fix/inspect forms are independent of each other. Each fix/inspect form can use the system to fetch the record, and all edits and updates occur using the master file MBF. This independent design has these major benefits:

- It organizes the application in a way that simplifies edits involving dependent fields across multiple forms.
- It enables consistent implementation of modeless processing for all master file applications and all forms within these applications.

Certain circumstances might justify deviation from this simple model. These circumstances are:

- Extremely large file formats

  When the number of columns in the master file plus the required control fields in the call data structure exceed technical limitations for data structures, the MBF can be split. You can split the MBF into one MBF that handles base data and performs all adds and deletes, and one or more MBFs that enable the calling program to update additional data when the base data has been established. In this case, it is usually logical to split it, regardless of the technical limitation. For example, assuming that the customer master file exceeded the data structure limitation, you would use these two MBFs to process the file:

- F0301ProcessMasterData
- F0301ProcessBillingData

  In this example, the F0301ProcessMasterData function processes the base data, and the F0301ProcessBillingData function updates additional data.

- Subordinate detail files

  Information can exist in addition to the primary master file that has been normalized to enable for a one-to-many relationship. Designing the Master File MBF strictly on the basis of how the database is designed translates into three calls. Including at least one occurrence of a detail relationship in the data structure of a Master File MBF is valid. This inclusion enables users to establish reasonably complete master file information using a simple interface to meet simple needs. Street addresses and phone numbers within Address Book are a good example. Customers expect that they can create an address book record by calling a simple address book API with basic identifying information, the street address, and a phone number.

MBF Information Structure

This section discusses the parameters of the MBF information structure.

Standard Parameters for Single-Record Master Business Functions

This table describes the standard parameters for single-record MBFs:

Name	Alias	I/O	Required /Optional	Description
Action Code	ACTN	I	Required	A = Add. I = Inquiry. C = Change. D = Delete. S = Same as except (the record is the same except for what the user changes).
Update Master File	EV01	I	Optional	0 = No update; edit only (default). 1 = Update performed.
Process Edits	EV02	I	Optional	1 = All Edits (default). 2 = Partial Edits (no data dictionary (DD)).
Suppress Error Messages	SUPPS	I	Optional	1 = Error messages are suppressed. 0 = Process errors normally (default).
Error Message ID	DTAI	O	Optional	Returns error code.
Version	VERS	I	Future	The default value is XJDE0001.

Application-Specific Control Parameters (Example: Address Book)

This table describes the application-specific parameters for Address Book:

Name	Alias	I/O	Required /Optional	Description
Address Book Number	AN8	I/O	Optional	For additions, AN8 is optional. For all other action codes, this parameter is required.
Same as except	AN8	I	Optional	Required for S = Action Code. The record is the same except for what the user changes.

Application Parameters (Example: Address Book)

This table describes the application parameters for Address Book:

Name	Alias	I/O	Required/Optional
Alpha Name	ALPH	I/O	Required
Long Address Number	ALKY	I/O	Optional
Search Type	AT1	I	Required
Mailing Name	MLMN	I	Required

Name	Alias	I/O	Required/Optional
Address Line 1	ADD1	I	Optional
City	CTY1	I	Optional
State	ADDS	I	Optional
Postal Code	ADDZ	I	Optional

Master Business Function Impact on Performance

Performance issues might occur regardless of how you handle large-format tables. Two options for improving performance are:

- Group data logically to enable data structures to be smaller and easier for the user to implement.

  This configuration does, however, force the user to make multiple calls to add or update an entire record in a table.
- Use a data structure that enables 300 fields.

  This configuration is cumbersome to implement, and the user can choose not to apply all of the fields.

Through different interfaces, the user can add additional data later. Most processes dictate that part of the data be added immediately, while related data can be added later. For example, the user might define a customer master record but wait until a later date to define the customer's billing instructions. Therefore, you should select the first option of splitting MBFs so that one MBF handles base data and one MBF handles additional data.

Working with Business Functions

Every business function must follow a defined structure and form. Every line of code must conform to the JD Edwards EnterpriseOne business function programming standards. Creating a business function involves these overall tasks:

- Use JD Edwards EnterpriseOne Object Management Workbench (OMW) to build business function data structures.
- Use OMW to build business function source and header files.
- Build and add type definitions for data structures to the header file.

Business function DLLs are consolidated. Therefore, you need to build each of the custom business functions into a custom DLL that you create. This process ensures that the custom business functions remain separate from JD Edwards EnterpriseOne business functions. The build program reviews the F9860 table to verify that the custom DLL exists.

When you create a custom business function, you need to specify one of the custom DLLs. If you do not, the build process builds the custom business function into the JD Edwards EnterpriseOne CCUSTOM.DLL, where CCUSTOM is the seven-character name of the company, which is the default.

Prerequisite

Create a data structure.

Creating a Custom DLL

To create a custom DLL :

1. In OMW, create a new Business Function Library.
2. In Windows, run BusBuild.exe.

   Typically, this file is located in ..\B9\System\Bin32\.
3. Rebuild all libraries by selecting Build, Rebuild Libraries in OMW.

   This process takes several minutes.

Specifying a Custom DLL for a Custom Business Function

To specify a custom DLL for a custom business function :

1. In Business Function Design Aid, enter the custom DLL name in the Parent DLL field.

   Note. You can also change the business function location if necessary.

   ---
2. Run the build for the business function.

Working with Business Function Builder

Use JD Edwards EnterpriseOne Business Function Builder to build business function code into a DLL. You can build C business functions, Named Event Rules (NERs), and table event rules. The process that occurs when you run JD Edwards EnterpriseOne Business Function Builder to build business functions includes compiling and linking. Compiling involves creating a business function object. Linking makes the object part of a DLL.

Note. Link All does not compile any business functions; it only links each DLL.

You usually use JD Edwards EnterpriseOne Business Function Builder to build a single business function. Whenever you create source code changes to a business function, you must build the business function to test it.

Build Output displays the results of the build. When the build is finished, the message `***Build Finished***` appears at the bottom of Build Output. The text after this line indicates whether the build was successful. If the build was successful, you can test the business function. Otherwise, you must correct any problems and rerun the build process.

The system creates a work directory when any object is built. This directory is in the destination directory that you specified, such as C:\b7\appl_pgf\work\buildlog.txt. This directory contains error and information logs. The build log contains the same information as the Build Output form in JD Edwards EnterpriseOne Business Function Builder.

Setting Build Options

Use options on the Build menu to control how and when the consolidated business function is built. This table describes the available options:

Option	Result
Build	Generates a makefile, compiles the selected business functions, and links the functions into the current consolidated DLL. Rebuilds only those components that are out of date.
Compile	Generates a makefile and compiles the selected business functions. The application does not link the functions into the current consolidated DLL.
ANSI Check	Reviews the selected business function for ANSI compatibility.
Link	Generates a makefile for each consolidated DLL and then builds each consolidated DLL. The application does not compile any of the selected business functions.
Link All	Generates a makefile for each consolidated DLL and then builds each consolidated DLL and links it to all business functions that are called. The application does not compile any of the selected business functions.
Rebuild Libraries	Rebuilds the consolidated DLL and static libraries from the .obj files.
Build All	Links and compiles all objects within each DLL.
Stop Build	Stops the build from finishing. The existing consolidated DLL remains intact.
Suppress Output	Limits the text that appears in Build Output.
Browse Info	Generates browse information when compiling business functions. Clear this option to expedite the build.
Precompiled Header	Creates a precompiled header when compiling a business function. When compiling multiple business functions, the Business Function Builder generally compiles faster if it uses a precompiled header.
Debug Info	Generates debug information when compiling. The Visual C++ can debug any function that was built with debug information. Clear this option to expedite the build.
Full Bind	Resolves all of the external runtime references for each JD Edwards EnterpriseOne consolidated DLL.

Reading Build Output

Build Output consists of a series of sections that display important information about the status of a build. You can use this information to determine whether the build completed successfully and to troubleshoot problems if errors occurred during the build.

Makefile Section

The makefile section indicates where Business Function Builder generated the makefile for a particular build. JD Edwards EnterpriseOne Business Function Builder generates one makefile for each DLL that it builds. A *Generating Makefile* statement should always appears for each DLL that you are building. If the makefile statement does not appear, then an error occurred. To resolve the error, you must complete these tasks:

- Verify that the local object directory exists.
- Verify that the permissions for the local object directory and the makefile are correct.

Begin DLL Section

Begin DLL indicates that Business Function Builder is building a particular DLL. For example, assume that the previous section begins with`*****CDIST*****`. A *Begin DLL* section appears for each DLL that you are building.

Compile Section

Before it build DLLs, Business Function Builder compiles the business functions in the DLLs first. The system displays a sequential list of each business function that the Business Function Builder attempts to compile. During the compilation process, these events might occur:

- Compiler Warning

  When a compiler warning occurs, JD Edwards EnterpriseOne Business Function Builder displays `warning` `C`*XXXX* (where *XXXX* is a number) and a brief description of the warning. To review information about the warning, search for the `C`*XXXX* value in Visual C++ online help. Warnings usually do not prevent the business function from compiling successfully. However, you can select the Warnings As Errors option in the Global Build form so that the business function will not build if any warnings occur.

- Compiler Error

  When a compiler error occurs, JD Edwards EnterpriseOne Business Function Builder displays `error` `C`*XXXX* (where *XXXX* is a number) and a brief description of the error. To review extended information about the error, search for the `C`*XXXX* value in Visual C++ online help. Because errors prevent the business function from compiling successfully, you must resolve them.

Link Section

After Business Function Builder has compiled the business functions for a DLL, it links them. This linking process creates the .lib and .dll files for the DLL. During linking, these events might occur:

- Linker Warning

  When a linker warning occurs, JD Edwards EnterpriseOne Business Function Builder displays `warning` `LNK`*XXXX* (where *XXXX* is a number) and a brief description of the warning. To review information about the warning, search for the `LNK`*XXXX* value in the Visual C++ helps. Warnings usually do not prevent the business function from linking successfully. You can select the Warnings As Errors option in the Global Build form so that the DLL will not build if it has any warnings occur.

- Linker Error

  When a linker error occurs, JD Edwards EnterpriseOnevBusiness Function Builder displays `error` `LNK`*XXXX* (where *XXXX* is a number) and a brief description of the error. To review extended information about the error, search for the `LNK`*XXXX* value in the Visual C++ helps. If a nonfatal error occurs, Business Function Builder still creates the DLL. However, JD Edwards EnterpriseOne Business Function Builder notes that the DLL was built with errors. If a fatal error occurs, JD Edwards EnterpriseOne Business Function Builder does not build the DLL.

Rebase Section

The Rebase Section displays information about rebasing. Rebase fine-tunes the performance of DLLs so that they load faster. Rebase does this by changing the desired load address for the DLL so that the system loader does not have to relocate the image. The system automatically reads the entire DLL and also updates fixes, debug information, checksum information, and time stamp values.

Summary Section

The Summary Section contains the most important information about the build. This section indicates whether the build is successful. The summary section begins with *****Build Finished*****. JD Edwards EnterpriseOne Business Function Builder also displays a summary report for each DLL that you attempted to build. This report includes this information:

- The number of warnings.
- The number of errors.
- Whether the DLL build is successful.

Building All Business Functions

You can use Build All to build all business functions. Build All performs the same operations as global link, and it recompiles all of the objects within each DLL. A system administrator usually runs Build All. Build All processes can take a long time. To run Build All, you must access BusBuild.

To build all business functions:

1. In Windows, run BusBuild.exe.

   Typically, this file is located in ..\B9\system\Bin32\.
2. In BusBuild, start the mass build by selecting Build, Build All.
3. Select one of these options for Build Mode:
   - *Debug*

     A build that includes debug information. After you perform a build, you can debug the built business function using the Visual C debugger.
   - *Optimize*

     A build that does not include debug information. Optimized builds generally cannot be debugged using the Visual C debugger.
   - *Performance Build*

     A build that is the same as an optimized build except that it includes information that helps developers measure the performance of business functions. Only JD Edwards developers should select this option.
4. Complete the Source Directory field.

   Use this field to specify where the business function source resides. Business function source includes all .c, .h, named event rules, and table event rules. Full packages usually have all business function sources. These are the options for location:
   - *Local*

     All business function source is on the local machine.
   - *Path Code*

     All business function source is in the path specified by the selected path code.
   - *Package*

     The All business function source is in the path specified by the selected package. If a package is built correctly, it typically contains all required business function sources. Generally, you should use *Package* for the location.

- *Pick Directory*

  All business function source is stored in another directory on the file server. You specify the directory.

5. Complete the Foundation Directory field.

   Use this field to specify the foundation to use for this build. The foundation that you select is the foundation on which you expect these business functions to run. These are the options for this field:

   - *Local*

     The recommended foundation is the local JD Edwards EnterpriseOne foundation.

   - *Foundation*

     The foundation table lists all registered JD Edwards EnterpriseOne foundations. Select a foundation from this table.

   - *Pick Directory*

     The JD Edwards EnterpriseOne foundation exists in a directory on the file server. You specify the directory. JD Edwards EnterpriseOne recommends this location.

6. Complete the Output Destination Directory field.

   Use this field to specify the location for the output of the build. The build output includes the file types: DLL, .LIB, .OBJ, and LOG. The location options are the same as those for *Source Directory*. Generally, you should select *Package* because it is a more stable snapshot of business function source.

7. Select any of these options:

   - Treat Warnings As Errors

     If you select this option, JD Edwards EnterpriseOne Business Function Builder does not build a business function if it encounters any warnings.

   - Clear Output Destination Before Build

     If you select this option, JD Edwards EnterpriseOne Business Function Builder deletes the contents of the bin32, lib32 and obj output directories before it builds all business functions.

   - Select Which DLLs to Build

     If you clear this option, JD Edwards EnterpriseOne Business Function Builder builds all DLLs. If you select this option, you can click the Select button and select which business function DLLs you want to build. Select this option if you want to build one or two DLLs. If you build only a subset of all DLLs, verify that the Clear Output Destination Before Build option is cleared.

   - Stop Level

     You can select the error level at which the build stops. You can ignore errors if you want to continue building despite them. You can specify that the build process stop if a DLL contains errors. You can stop on the first compile error.

   - Generate Missing Source Report

     If you select this option, v Business Function Builder generates a report in the work directory of the destination. This report is called NoSource.txt. It contains business function source file names that do not have a .c file but do have a record in the F9860 table. To resolve the information in this report, you can produce the correct .c file for the business function, or you can delete the source file from the F9860 table. It is recommended that you select this option.

   - Generate ER Source

     If you select this option, v Business Function Builder generates NER and table event rule source before building business functions.

- Verify Check-in

  If you select this option, the system builds only objects checked in to a specified path code. A log file, Notchkdn.txt, is written to the same directory as Nosource.txt. Objects that are not checked in to the path code will be listed in this log and in Buildlog.txt.

Select the From RDB option to generate work from any path code. If this option is cleared, the business function builder assumes that the event rules source can be generated from the source directory specification files.

If you are troubleshooting a build initiated by *Package Build*, then the previous settings should already be set to the correct values. In this case, click Build to rebuild the problem DLLs.

Note. You can also run this build by selecting the Build BSFN option on in a package build.

Using the Utility Programs

The Tools menu contains several utility programs that assist in the build process. This table lists those utilities:

Utility	Purpose
Synchronize JDEBLC	You run the Synchronize JDEBLC program to reorganize JD Edwards EnterpriseOne business functions into new DLL groupings. This program synchronizes DLL field for the local JDEBLC parent specification table with the parent DLL in the F9860 table. Use this program with caution. You typically use this program only if you have manually dragged business function DLLs from a recent package build and you are experiencing failures in the business function load library.
Dumpbin	You run the Dumpbin program to verify whether a particular business function built successfully. This program displays all the business functions that were built into the selected consolidated DLL.
PDB (Program DeBug file) Scan	You receive a CVPACK fatal error when one of the object files that you are trying to link is incorrectly compiled with PDB information. To resolve this problem, you can use the PDB Scan to identify any object fields that were built with PDB information. Recompile any business functions that the PDB Scan reports.
Customize	You use Customize to add programs to the Tools menu. For example, you could add the programming tool and pass that tool a file name as a parameter when it opens.
Safety Check	You use Safety Check to check selected files (.c, .h or both) for: • global variables • static variables • extern declarations • non-”threadsafe” ANSI C APIs
Safety Check-Check All	You use Safety Check-Check All to check all files (.c, .h or both) in a directory for the same conditions as for Safety Check.

Resolving Errors with JDEBLC, Dumpbin, and PDB

You use JD Edwards EnterpriseOne Business Function Builder tools to help you resolve errors. If you notice any unresolved external errors during a business function build, the consolidated DLL still builds, and the software should run normally. However, it cannot execute any unresolved business function.

Use the dumpbin tool to verify that a particular business function is present in a consolidated DLL. If a business function is present, its name appears in the dumpbin output, followed by a nonzero number in parentheses.

Use the PDB scan to resolve the CVPACK fatal error. The CVPACK error occurs when the Business Function Builder attempts to link an object file that was built with PDB (Program DeBug file) information. The PDB scan finds the problem object file. You must then recompile the problem object file on the machine with the JD Edwards EnterpriseOne Business Function Builder.

If a business function is compiled using Visual C++, it will not work properly. You can use PDB scan to identify any business functions that have been built outside of JD Edwards EnterpriseOne Business Function Builder. Use JD Edwards EnterpriseOne Business Function Builder to rebuild these functions so that they work properly.

If one of the DLLs is out of synch, you must rebuild it using the Build option. This generates a makefile and then relinks all the business functions within it.

The Synchronize JDEBLC option from the JD Edwards EnterpriseOne Business Function Builder Tools menu corrects any misplaced or incorrectly-built business functions. This option reviews the server DLLs and determines whether the local workstation specifications match those of the server. If they do not, then JD Edwards EnterpriseOne Business Function Builder will rebuild the business functions in the correct DLL on the server and relink them.

The Build Log contains these sections:

Section	Description
Build Header	This section defines the configuration for a specific build, including the source path, foundation path, and destination path.
Build Messages	This section displays the compile and link activity. During a compile, a line is output for each business function that was compiled Any compile errors are reported as error c*xxxx*. During the link part, business function builder outputs the text `Creating library ....` This text might be followed by linker warnings or errors.
Build Summary	The last section of the build summarizes the build for each DLL. This summary is in the form `x error(s), x warnings (y).` The summary indicates the status of the build. If you have no warnings and no errors, then the build was successful. If the summary reports an error, search the log for the word *error* to determine the source of the error. Typical build errors are syntax errors and missing files.

Customizing the Tools Menu

This table lists the sections of the Customize menu option:

Menu Option	Usage
Menu Contents	Review all current tools menu customizations.
Menu Text	Enter the text to display in the menu.
Command	Enter the executable to run. You must supply a full path for any program that does not reside in `system\bin32` or that is not defined in Initial Directory.
Arguments	Specify any command line arguments to pass to the executable.

Menu Option	Usage
Initial Directory	Specify the initial directory that should be used by the executable, if it is not `system\bin32.`
Include in Build	Select to display output from the program as part of the build process. **Note.** This option is only valid and will only appear for Release 8.11 SP1 or later. If you are running an earlier version, this option is not available, and Safety Check does not run during build. You must, instead, run Safety Check manually from the menu.
Hide Window	Select to hide command windows. The functionality remains the same.

This table lists the buttons in the Customize menu option:

Button	Usage
Add	Select to enter new programs to appear in the pull-down menu
Remove	Select to remove the selected item from the menu
Move Up	Select to move the selected item up in the menu
Move Down	Select to move the selected item down in the menu
Ellipsis	Select to open a file or directory dialog so that you can browse for a file or directory
Question Mark	Select to display a list of substitutions you can use as part of command line arguments. In our SafetyCheck example, one of the command line arguments is: `--F <source_file>`. By specifying `<source_file>`, you are telling SafetyCheck to use as its input file the selected source file. When BusBuild starts the build process, it can determine which file is being built and substitute that name in place of the text`<source_file>.`

Threadsafe Code

Before this release, the **jdeCallObject** kernel was a single-threaded entity. This meant only one application business function per kernel could be executing at a time. JD Edwards EnterpriseOne business functions were therefore developed with this limitation in mind. From this release forward, though, both existing functions and future design methodologies have to change. The **jdeCallObject** kernel is now multi-threaded, which enables multiple application BSFNs to run simultaneously.

Business functions that are not threadsafe will run unpredictably in this new environment. To be considered threadsafe, BSFNs cannot use:

- Global variables.
- Static variables.
- External declarations.
- Non-threadsafe ANSI C APIs.

Safety Check is a source code analysis tool that scans C source code and header files for non-threadsafe behaviors. Given a source or header file, Safety Check finds all instances of non-threadsafe code, returning line numbers and code fragments.

Several non-threadsafe APIs have a JD Edwards EnterpriseOne replacement. These replacement APIs have the same parameters as the non-threadsafe C APIs, except where noted. Most non-threadsafe APIs do not have a JD Edwards EnterpriseOne replacement. These APIs and their replacements do not necessarily have the same parameters. Use care when using these APIs.

Note. Only those APIs that differ from the standard C APIs are documented in this Guide.

This table lists the non-threadsafe C APIs for which SafetyCheck searches, the threadsafe standard C replacements, and the threadsafe JD Edwards EnterpriseOne replacements (if applicable):

Non—Threadsafe Standard C API	Threadsafe Standard C API	Threadsafe JD Edwards EnterpriseOne API
acltostr	**acltostr_r**	None
asctime	**asctime_r**	**jdeJAsctime**
crypt	**crypt_r**	None
ctime	**ctime_r**	**jdeJCtime**
drand48	**drand48_r**	None
ecvt	**ecvt_r**	None
encrypt	**encrypt_r**	None
endgrent	**endgrent_r**	None
endhostent	**endhostent_r**	None
endnetent	**endnetent_r**	None
endprotoent	**endprotoent_r**	None
endpwent	**endpwent_r**	None
endservent	**endservent_r**	None
endspwent	**endspwent_r**	None
endusershell	**endusershell_r**	None
endutent	**endutent_r**	None
erand48	**erand48_r**	None
fcvt	**fcvt_r**	None
fgetgrent	**fgetgrent_r**	None
fgetpwent	**fgetpwent_r**	None
getdate	**getdate_r**	None

Non—Threadsafe Standard C API	Threadsafe Standard C API	Threadsafe JD Edwards EnterpriseOne API
getdiskbyname	**getdiskbyname_r**	None
getgrent	**getgrent_r**	None
getgrgid	**getgrgid_r**	None
getgrnam	**getgrnam_r**	None
gethostbyaddr	**gethostbyaddr_r**	**jdeGetHostByAddr_r**
gethostbyname	**gethostbyname_r**	**jdeGetHostByName_r**
gethostent	**gethostent_r**	None
getlocale	**getlocale_r**	None
getlogin	**getlogin_r**	None
getnmtent	**getmntent_r**	None
getnetbyaddr	**getnetbyaddr_r**	None
getnetbyname	**getnetbyname_r**	None
getnetent	**getnetent_r**	None
getprotobyname	**getprotobyname_r**	**jdeGetProtoByName_r**
getprotobynumber	**getprotobynumber_r**	None
getprotoent	**getprotoent_r**	None
getpwent	**getpwent_r**	None
getpwnam	**getpwnam_r**	None
getpwuid	**getpwuid_r**	None
getservbyname	**getservbyname_r**	None
getservbyport	**getservbyport_r**	None
getservent	**getservent_r**	None
getspwaid	**getspwaid_r**	None
getspwnam	**getspwnam_r**	None
getspwuid	**getspwuid_r**	None
getusershell	**getusershell_r**	None

Non—Threadsafe Standard C API	Threadsafe Standard C API	Threadsafe JD Edwards EnterpriseOne API
getutent	**getutent_r**	None
getutid	**getutid_r**	None
getutline	**getutline_r**	None
gmtime	**gmtime_r**	**jdeGmtime**
inet_ntoa	**inet_ntoa_r**	**jde_inet_ntoa_r**
jrand48	**jrand48_r**	None
l64a	**l64a_r**	None
lcong48	**lcong48_r**	None
localtime	**localtime_r**	**jdeLocaltime** **Note.** The parameters changed on this due to the need to send a location to store the value. The standard C call stores it in a global static variable, which is not threadsafe.
lrand48	**lrand48_r**	None
ltoa	**ltoa_r**	None
ltostr	**ltostr_r**	None
mrand48	**mrand48_r**	None
nrand48	**nrand48_r**	None
ptsname	**ptsname_r**	None
pututline	**pututline_r**	None
rand	**rand_r**	**jdePPRand** **Note.** Must be used in conjunction with **jdePPSRand** to seed the random number generator correctly. Existing calls to **srand** should be replaced with **jdePPSRand.**
readdir	**readdir_r**	None
seed48	**seed48_r**	None
setgrent	**setgrent_r**	None
sethostent	**sethostent_r**	None
setkey	**setkey_r**	None
setlocale	**setlocale_r**	**jdeSetLocale**

Non—Threadsafe Standard C API	Threadsafe Standard C API	Threadsafe JD Edwards EnterpriseOne API
setnetent	**setnetent_r**	None
setprotoent	**setprotoent_r**	None
setpwent	**setpwent_r**	None
setservent	**setservent_r**	None
setspwent	**setspwent_r**	None
setusershell	**setusershell_r**	None
setutent	**setutent_r**	None
srand	**srand_r**	**jdePPSRand**
srand48	**srand48_r**	None
strerror	**strerror_r**	None
strtoacl	**strtoacl_r**	None
strtoaclpatt	**strtoaclpatt_r**	None
strtok	**strtok_r**	None
ttyname	**ttyname_r**	None
ultoa	**ultoa_r**	None
ultostr	**ultostr_r**	None
utmpname	**utmpname_r**	None
wcstok	**wcstok_r**	None

See Appendix A, “JD Edwards EnterpriseOne APIs,” JD Edwards EnterpriseOne Threadsafe APIs, page 160.

Safety Check Usage

During the course of development, there may be times when a non-threadsafe type of code must be used. You can mark source code with an explanation about why the non-threadsafe code exists. Safety Check will then display this information as part of its run. To mark source code with an exception, include a comment in this format: `/*_LRBF <comment text */`. The comment must begin with "`/*_LRBF.`" The remainder of the comment can span multiple lines and include any other necessary text. The entire comment will print as part of Safety Check output.

You control Safety Check functionality through several options, at least one of which must be supplied. Multiple options are supported. Quotation marks are required only when the path specified contains spaces. For example, if the single C source file b1234.c is stored in the "c:\source" directory, you could call SafetyCheck in one of two ways: SafetyCheck --F c:\source\b1234.c or SafetyCheck --F "c:\source\b1234.c" However, if the same C source file is stored in the "c:\test files", you must enclose the path/filename in quotations: SafetyCheck --F "c:\test files\b1234.c"

Argument	Usage
--F *<C source file>*	Use to check a single C source file, for example, **--F** c:\test\b1234.c
--I *<Header file>*	Use to check a single header file, for example, **--I** c:\include\b1234.h
--FD *<C source directory>*	Use to check all C source files in a given directory, for example, --FD c:\my project\source. **Note.** Do not include a trailing slash as part of the directory argument.
--ID *<Header file directory>*	Use to check all header files in a given directory, for example **--ID** c:\my project\include **Note.** Do not include a trailing slash as part of the directory argument.

--P *<Project file>*	Use to create a text file that contains a list of files, each of which will be scanned by Safety Check. The project file should contain multiple lines of the form: **SOURCE=***<fully qualified file name>*
	
	Note. Do not use quotation marks in the project file.
	
	For example, a project file that specifies three files to scan could look like this:
	
	`SOURCE=c:\my project\source\b1111.c`
	
	`SOURCE=c:\my project\source\b2222.c`
	
	`SOURCE=c:\my project\include\main.h`

Argument	Usage
–csv	Use to produce output in a comma-delimited format. The output will contain these elements: • File (the fully qualified file name) • Line (the line number of the erroneous code) • Global (1 if a global was found, 0 if not.) • Static (1 if a static was found, 0 if not.) • Extern (1 if an external declaration was found, 0 if not.) • API (1 if a non-threadsafe API was found, 0 if not.) • BraceMismatch (1 if scanning could not complete due to a brace mismatch) • Exception (1 if an exception comment was found, 0 if not.) • CouldNotOpen (1 if the file could not be opened, 0 if it could.) • NotCSource (1 if the file name did not end in either ".c" or ".h") • C++Comment (1 if a C++ style comment was found) • CapInclude (1 if a capital letter was used in a #include) • LastChar (1 if the last character was not a new line character) • CommentInComment (1 if a comment was found inside a comment)
–X	Select to print a warning message when a file to check is specified that does not end in "c" or "h". By default, these warning messages are hidden.

Safety Check Output

A “clean” Safety Check run will produce output of this format:

```
---------------- SafetyCheck Started ---------------

Scanning d:\safetychecktestrun\source\b03b0011.c...

---------------------- Done ----------------------

1 Files Processed  0 Errors  0 Warnings
```

“Files processed” indicates how many files were scanned. “Errors" reports the number of file-based errors encountered. “Warnings” reports the number of problems found while scanning the specified files.

A “dirty” Safety Check run will produce output of this format:

```
---------------- SafetyCheck Started ---------------

Scanning d:\safetychecktestrun\source\b03b0011.c...

d:\safetychecktestrun\source\b03b0011.c(186): Global variable found

int iGlobal = 0;

---------------------- Done ----------------------

1 Files Processed  0 Errors  1 Warnings
```


In this case, the output indicates:

- A problem was found in d:\safetychecktestrun\source\b03b0011.c
- The problem occurred on line 186.
- The problem found was the presence of a global variable.
- The section of code that caused the problem is “`int iGlobal = 0;`”

Note that the global variable was specified as a "Warning" and not an "Error".

Safety Check Limitations

1. Safety Check is a static code analysis tool that does not perform preprocessing of source code. Therefore, macro substitutions may introduce non-threadsafe behaviors that cannot be detected by Safety Check.
2. Safety Check does not know which compile-time flags may be set. Problems will occur in code that looks like this because the number of open braces does not match the number of close braces:

```
int FunctionOne(int i)
{
 if (i == 0)
 #ifdef FLAG1
  {
   ++i;
 #else
  {
   --i;
 #endif
  }

}
```

3. Non-threadsafe code may still exist even though Safety Check reports no warnings. Safety Check is looking for the presence of only four specific code elements (globals, variables, externs and non-threadsafe ANSI C APIs). Do not rely solely on a “clean” run of Safety Check as the only test of whether the code is threadsafe.

Understanding Business Function Processing Failovers

In some instances in which a business function fails to process correctly, the software can attempt to recover and reprocess the transaction. The system recognizes two principle failure states: process failure and system failure.

A process failure occurs when a jdenet_k process aborts abnormally. For a process failure, the software server processing launches a new jdenet_k process and continues processing.

A system failure occurs when all the server processing fails, the machine itself is down, or the client cannot reach the server because of network problems. For a system failure, business function processing must be rerouted either to a secondary server or to the local client. The system uses this process to attempt to recover from this state:

- When the call to the server fails, the system attempts to reconnect to the server.
- If reconnect succeeds and no cache exists, the system reruns the business function on the server.

  If a cache does exist, the system forces the user out of the application.
- If reconnect fails and no cache exists, the system switches to a secondary server or to the local client.

  If a cache does exist, the system forces the user out of the application.

After one module switches, all subsequent modules switch to the new location.

Working with Business Function Documentation

This section provides an overview of business function documentation, and discusses how to:

- Create business function documentation.
- View documentation from the Business Function Documentation Viewer.

Understanding Business Function Documentation

Business function documentation explains what individual business functions do and how they should be used. The documentation for a business function should include this type of information:

- Purpose.
- Parameters (the data structure used).
- Descriptions for each parameter that indicate required input and output, and explain return values.
- Related tables (the table accessed).
- Related business functions (business functions called from within the function itself).
- Special handling instructions.

You use Business Function Design and Data Structure Design to document the business functions.

Creating Business Function Documentation

You can create business function documentation for several levels, including these:

- Business Function Notes

  Documentation for the specific business function that you are using.
- Data Structure Notes

  Notes about the data structure for the business function.
- Parameter Notes

  Notes about the actual parameters in the data structure.

Generating business function documentation provides you with an online list of information about business functions that you can view through the Business Function Documentation Viewer (P98ABSFN). Typically, the system administrator performs this task because generating the business function documentation for all business functions takes considerable time. If you create new business function documentation, you need to regenerate the business function documentation for that business function only.

Run UBE R98ABSFN, batch version XJDE0001 to generate all business function documentation. The system creates a hypertext markup language (HTML) link for each business function for which you generated documentation. It also creates an Index HTML file. These HTML files appear in the output queue directory.

Viewing Documentation from Business Function Documentation Viewer

You can use Business Function Documentation Viewer to view documentation for all business functions or selected business functions. After you generate the report, use the Business Function Documentation Viewer (P98ABSFN) to display the information. It is suggested that you use this method to view business function documentation.

The Business Function Documentation form contains the HTML index that you generated. To view the entire index or select specific functions, click the appropriate letter in the index. Double-click a business function to view documentation that is specific to that function.

The media object loads the HTML index of the business functions based on a media object queue. In the media object queue table, a queue named Business Function Doc is defined.

This queue must point to the directory in which the business function HTML files are located. The system administrator usually generates the documentation for all business functions. Because the generation process places the documentation files in the local directory, the administrator must then copy the files to a central directory on the deployment server. The files must be copied to the media object queue for media object business function notes. If you are using the standalone version of the software, this path is usually the output directory from the Network Queue Settings section of the jde.ini file. If this entry is not in the jde.ini file, it is in the print queue directory in the JD Edwards EnterpriseOne software directory.

CHAPTER 4

Understanding Record Locking

This chapter provides overviews of record locking, optimistic locking, and pessimistic locking.

Record Locking

JD Edwards EnterpriseOne does not implement any record-locking techniques. It relies on the native locking strategy of the vendor database management system.

In specific situations, the vendor database does not automatically lock as needed. In these situations, you can instruct JD Edwards EnterpriseOne to control record locking. For example, you can mandate record locking on the Next Numbers table to ensure the integrity of the Next Numbers feature.

You can lock JD Edwards EnterpriseOne records using one of the following methods:

- Optimistic locking

  Use optimistic locking (sometimes referred to as record change detection) to prevent a user from updating a record if it has changed between the time the user inquired on the record and the time user updates the record.

- Pessimistic locking

  Use pessimistic locking to prevent attempts to update the same record at the same time by different applications or users. The record is locked before it is updated.

Optimistic Locking

You can set optimistic locking in the workstation jde.ini file. This type of database locking prevents a user from updating a record that changed since the user has inquired about it. If the record has changed, the user must select the record again and then make the change. This feature is available for business functions, table I/O, and Named Event Rules.

For example, assume that two users are working in the Address Book application The following table illustrates the optimistic locking process:

Time	Action
10:00	User A selects Address Book record 1001 to inspect it.
10:05	User B selects Address Book record 1001 to inspect it. Both users now have Address Book record 1001 open.

Time	Action
10:10	User B updates a field in Address Book record 1001 and clicks OK. JD Edwards EnterpriseOne updates Address Book record 1001 with the information entered by User B.
10:15	User A updates a field in Address Book record 1001 and clicks OK. JD Edwards EnterpriseOne does not update Address Book record 1001, and the system displays a message informing User A that the record has changed during the time that User A was viewing it. For User A to change the record, User A must re-select it and then update it.

When the system detects that a record change has occurred, it displays a message indicating that the record has been changed since it was retrieved.

Pessimistic Locking

Pessimistic locking is sometimes referred to as record locking. You can use pessimistic locking to prevent multiple users or applications from updating the same record at the same time. For example, suppose a user enters a transaction that uses Next Numbers. When the user clicks OK, the Next Numbers feature selects the appropriate Next Numbers record, verifies that this number is not already in the transaction file, and then updates the Next Numbers record by incrementing the number. If another process tries to access the same Next Numbers record before the first process has successfully updated the record, the Next Numbers function waits until the record is unlocked and then completes the second process.

Pessimistic locking in JD Edwards EnterpriseOne is implemented by calling published JDEBase APIs. When you use pessimistic locking, you should consider the time required to select and update a record because the record is locked until the update is complete. Transaction processing uses a special set of locking APIs. A locked record might or might not be part of a transaction. Record locking APIs are independent of the transaction and its boundaries. They always lock, regardless of whether you are in manual or auto commit mode.

Records that are updated using pessimistic locking APIs (such as JDB_FetchForUpdate or JDB_UpdateCurrent) within a transaction boundary are locked from the time the record is selected for update until the commit or rollback occurs. Records within the transaction boundary that are updated without using pessimistic locking APIs are locked from the time of the update until the commit or rollback occurs. This is also true if you use a business function to define and activate transaction processing.

Using Pessimistic Locking Within a Transaction Boundary

You might need to use pessimistic locking in conjunction with transaction processing. For example, if you want the system to lock records between the read operation and the update, you must use pessimistic locking.

Business Functions and Pessimistic Locking

You might want to use pessimistic locking in a business function if the business function updates a table. The table being updated should have a high potential for record contention with another user or job. Remember that you should lock records for as short a time as possible. Ensure that the select or fetch for an update occurs as closely to the update as possible.

CHAPTER 5

Debugging Business Functions

This chapter provides overviews of debugging, the debugging process, interpreted vs. compiled code, debugging strategies, and the debug log, and discusses how to debug business functions with Microsoft Visual C++.

Debugging

Debugging is the method you use to determine the state of your program at any point of execution. Use debugging to help you solve problems and to test and confirm program execution.

Use a debugger to stop program execution so you can see the state of the program at a specific point. This enables you to view the values of input parameters, output parameters, and variables at the specified point. When program execution is stopped, you can review the code line-by-line to check such issues as flow of execution and data integrity.

You use the Visual C++ Debugger to debug C business functions.

Debugging Strategies

You can use several strategies to make debugging faster and easier. Begin by observing the nature of the problem.

Is the Program Ending Unexpectedly?

If the program is ending unexpectedly, the cause is likely an unhandled exception. An unhandled exception is a failure to handle memory correctly. It is an easy problem to track down if it is happening in the same place: simply set breakpoints at strategic points throughout the code and run the program until you find the problem.

If other objects are missing, termination is more abrupt. Remember to transfer all Media Object (also called Generic Text) objects correctly. If an application has a Row exit to an application that does not exist, an unhandled exception in the program occurs immediately.

Termination of the program is more abrupt and less helpful when other kinds of objects are missing. You must review all of the pieces of the application to verify that they are all present and correctly built. A common error is to overlook media objects. If you cannot enter the program at all, a missing object is most likely the problem.

Ensure that the program is terminating in the same place. If the program is failing to restore memory after its use, the program might eventually have insufficient memory to run. If so, you must reboot the workstation to restore memory.

Is the Output of the Program Incorrect?

Incorrect program output typically indicates a flaw within the logic of the code. To help find the error:

- Set a breakpoint in the code prior to the point where the bad output is produced.
- Step through the ER line by line, while monitoring the values of relevant ER variables.

  At some point, a variable will probably take on an erroneous value that subsequently produces incorrect output.
- If that point occurs before your breakpoint, set another breakpoint earlier in the code and restart the application.
- Continue this process until you find the statement that is causing the wrong value to be assigned to the variable.

Where Else Could the Problem Be Coming From?

Spend some time thinking about where the source of the problem might be. If you don't know which ER event is causing an error, try to isolate it. For example, you might be able to temporarily disable the ER one event at a time to see if the error still happens. You can try to repeat the processing of a single event by doing unnatural actions in the GUI, like toggling up and down between grid rows to force the execution of the **Row Is Exited** event. There are no predefined debugging strategies that will work in any given situation. Be creative and be persistent, until you narrow down the problem to its source.

Debug Logs

You can output to a file a log of SQL statements and events by changing the line in the jde.ini file under `[DEBUG]` from `Output = NONE` to `Output = FILE`, as in the following sample. This is a useful debugging tool when you have narrowed a problem to a specific issue involving the JDEDB APIs.

```
[DEBUG]
TAMMULTIUSERON=0
Output=FILE
ServerLog=0
LEVEL=BSFN,EVENTS
DebugFile=c:\jdedebug.log
JobFile=c:\jde.log
Frequency=10000
RepTrace=0
```

You can set breakpoints and examine the code.

Debugging Business Functions with Microsoft Visual C++

This section provides an overview of the Microsoft Visual C++ debugger and describes how to:

- Debug business functions attached to interactive applications.
- Use SQL log tracing.

- Use debug tracing.

Understanding the Visual C++ Debugger

You can use Microsoft Visual C++ to debug business functions that are written in C. You can debug business functions that are attached to interactive applications or to batch applications. The business function must be configured to run locally.

If you are debugging ER for business functions and C business functions, you can use the JD Edwards EnterpriseOne debugger and the Visual C++ debugger together. Follow the process until you log into JD Edwards EnterpriseOne. At that point, follow the steps for the JD Edwards EnterpriseOne debugger. Program execution stops if C code is accessed. You can then use Visual C++ to continue debugging. This method is useful if you are trying to locate a problem and are not sure whether the problem is in a C business function or in the application that calls the business function.

You must use the Microsoft Visual C++ Debugger to debug business functions that were written with the Event Rules scripting language and then interpreted as C code, or that were originally written in C . You can run the entire JD Edwards EnterpriseOne system through the Visual C++ debugger (that is, you can start the activeConsole.exe or JD Edwards Solution Explorer file from within the Visual C++ Debugger). This enables you to step out of the tool application code into the business functions that are called in the ER.

You can use the debugger to debug a C program and interactively stop and start it as needed. During debugging, you can check specific values of variables and parameters to determine whether a program is running correctly. You can also step through the code to see what code is actually being executed.

The debug commands are listed in the Debug menu. You can customize the tool bar to contain debug buttons, which you can use instead of the menu.

The Visual C++ has many features in the Debug menu. The Visual C++ debugger helps you efficiently solve real-world problems.

The Go Command

You can run a program using the Go command from the Debug menu. The program runs until completion unless you set up breakpoints.

The Step Command

The Step command is available on the Debug menu and executes the current line of code. When the line of code has been executed, the yellow arrow cursor appears on the next line of code to be executed.

The Step Into Command

You can access the Step Into command from the Debug menu. Use this command when the current line of code contains a function call. The debugger steps into the function so that it can be debugged line by line. When the function is complete, the debugger returns to the next line of code after the function call in the calling routine. If the source code of the function to be stepped into does not exist on the workstation, the debugger skips over the line of code as though the Step command was used.

Stepping into a standard C function takes you into the function, which you might not want to do. If so, use the Step Over command to skip those functions.

Setting Breakpoints

You use breakpoints to run the program until it reaches a certain line of code. If a breakpoint is set, the Go command runs the program until it encounters that line of code.

You can set a breakpoint by placing the cursor anywhere on the line of code. When you select Debug, Breakpoints, a red octagon appears to the left of the line of code where the breakpoint is set. When the program is run, all lines of code up to the breakpoint are executed. To continue execution after the breakpoint, you can use Step, Step Into, or Go.

Using Watch

You can use Watch to inspect what values variables are set to. To use Watch, click the item that you want to watch and drag it to the Watch window.

Locals Window

All local variables and parameters to a function are listed with their data types and values in the Locals window. You can modify the values of all items in the Locals window during debugging. This is useful if you are debugging infinite loops.

Understanding Visual C++ Debugger Tracing Utilities

Visual C++ has two tracing utilities that you might find valuable: SQL Log Tracing and debug tracing. You can use SQL Log Tracing to help you determine the exact SQL statement that is generated and sent to the database.

Debugging Business Functions Attached to Interactive Applications

To debug a business function attached to an interactive application:

1. Close the application.

   The application must be closed to debug in Visual C++.
2. Open Visual C++ and verify that all workspaces have been closed.
3. Select File, Open.
4. Select List Files of Type to accept executables (.exe).
5. Select activConsole.exe on path \b9\System\bin32 and click the OK button.

   The system creates a project workspace.
6. Select Project, Settings.
7. Click the Debug tab.
8. In the Category list, select Additional DLLs.
9. Click the Browse button to select the CALLBSFN.dll (which must be built in debug mode) or other appropriate DLL on path \b9*path*\bin32, where *path* varies, depending on the path code.
10. Click the OK button.
11. Select the .h and .c files for the source that you want to debug from and then select File, Open.
12. To set breakpoints in the code, select Edit, Breakpoints.

    If this message appears, click the OK button:

    ```
    cannot open *.pdb
    ```

    If a message appears notifying you that breakpoints have been moved to the next valid lines, a source code and object mismatch might exist, and you might need to rebuild the business function.

13. Select Build, Start Debug, Go.

    The JD Edwards EnterpriseOne sign-in window appears.

14. Sign in to the application as you normally would sign in.
15. Run the application.

    When the application reaches the business function in debug, the debugger opens or displays the C code in Visual C so that you can step through it.

Using SQL Log Tracing

This task is useful only for ODBC connections.

To use SQL Log tracing:

1. From the Control Panel on the workstation, select Administrative Tools, and then Data Sources (ODBC).
2. Select the 32 bit ODBC driver, and then click the Tracing tab.
3. Specify when you want the system to trace.
4. Specify the log output path in the Log file Path.

Using Debug Tracing

To use debug tracing:

1. In the jde.ini file under `[DEBUG]`, set `Output=FILE`.
2. Change the value for `Level=` to suit the specific debugging needs.

   Possible values for Level are contained in the comment line following the `Level=` line. Any combination is acceptable. Use commas to separate values.

APPENDIX A

JD Edwards EnterpriseOne APIs

This appendix describes the public APIs available for JD Edwards EnterpriseOne.

General APIs

This section discusses APIs that can used in a number of different JD Edwards EnterpriseOne systems.

jdeCreateGuid

Syntax

`void jdeCreateGuid(JDEGUIDBIN *pGuid)`

Description

jdeCreateGuid generates a GUID, also known as a UUID. A GUID is a 128 bits long, but is often represented by a 36 character string. This API generates the 128 bit, binary representation.

Parameters

Parameter	Description
pGuid	Destination pointer for generated GUID.

Example

```
JDEGUIDBIN zGUID = {0};
 jdeCreateGuid(&zGUID);
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeCreateGuidString, page 95
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGuidCompare, page 96
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGuidToString, page 97

jdeCreateGuidString

Syntax

`void jdeCreateGuidString(JDEGUIDSTR szGuid)`

Description

jdeCreateGuidString generates a GUID (Globally Unique IDentifier) also known as a UUID (Universally Unique IDentifier) and returns the string representation. A GUID is a 128 bits long, but is often represented by a 36 character string. This API generates the string representation, such as 4C32C69E-4D33-11D8-B299-72E2054054E5.

Parameters

Parameter	Description
JDEGUIDSTR *szGuid*	Destination pointer for generated GUID string.

Example

```
JDEGUIDSTR szGUID = {0};

 jdeCreateGuidString(szGUID);

 jdePrintf(_J("Generated GUID: %ls\n"), szGUID);
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeCreateGuid, page 95
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGuidCompare, page 96
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGuidToString, page 97

jdeGuidCompare

Syntax

```
int jdeGuidCompare(
        JDEGUIDBIN *pG1,
        JDEGUIDBIN *pG2
        )
```

Description

The GUIDs (Globally Unique IDentifier) generated by jdeCreateGuid can be ordered lexicographically. This API provides a lexicographical comparison of two GUIDs.

Parameters

Parameter	Description
JDEGUIDBIN **pG1*	First GUID to compare.
JDEGUIDBIN **pG2*	Second GUID to compare.

Returns

An integer. The values are:

Value	Description
1	*pG1 > pG2*
0	*pG1* and *pG2* point to identical GUID representations.
-1	*pG1 < pG2*

Example

```
JDEGUIDBIN zGUID1 = {0};
JDEGUIDBIN zGUID2 = {0};
...
jdeCreateGuid(&zGUID1);
jdeCreateGuid(&zGUID2);
...
if (jdeGuidCompare(&zGUID1, &zGUID2) == 0)
{
jdePrintf(_J("GUIDs are equivalent\n"));
}
...
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeCreateGuid, page 95
Appendix A, "JD Edwards EnterpriseOne APIs," jdeCreateGuidString, page 95
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGuidToString, page 97

jdeGuidToString

Syntax

```
void jdeGuidToString(
      JDEGUIDSTR szDestStr,
      JDEGUIDBIN zGuid
      )
```

Description

jdeCreateGuid generates a 36-character string representation for the 128-bit binary representation of a GUID (Globally Unique IDentifier) that is passed in to it. This is an example of a GUID: 4C32C69E-4D33-11D8-B299-72E2054054E5.

Parameters

Parameter	Description
JDEGUIDSTR *szDestStr*	Destination pointer for GUID string.
JDEGUIDBIN *zGuid*	Structure containing source GUID.

Example

```
JDEGUIDBIN zGUID1 = {0};
JDEGUIDSTR szGUID2 = {0};
...
jdeCreateGuid(&zGUID1);
...
jdeGuidToString(szGUID2, zGUID1);
...
jdePrintf(_J("String representation: %ls\n"), szGUID2);
...
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeCreateGuid, page 95
Appendix A, "JD Edwards EnterpriseOne APIs," jdeCreateGuidString, page 95
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGuidCompare, page 96

jdeEncryptWKey

Syntax

```
int jdeEncryptWKey(
     BYTE *outbuf,
     int *poutlen,
     JCHAR *inbuf,
     int inlen,
     JCHAR *encryptkey,
     int keylen,
     int type
     )
```

Description

jdeEncryptWKey performs encryption with input JCHAR string data and JCHAR encrypt key string, and then returns encrypted bytes and length. The maximum length of JCHAR encrypt key is 16, and the minimum length of JCHAR encrypt key is 4. A TripleDES algorithm is implemented currently.

Parameters

Parameter	Description
outbuf	Output. The number of encrypted bytes. If NULL, this API will fail.
poutlen	Output. The length of encrypted bytes.
inbuf	Input. JCHAR string data. If *inbuf* string data is NULL or has a length of zero, this API will fail. However, if *inbuf* is a blank space string with length greater than zero, the API will encrypt the string as a valid JCHAR string.
inlen	Input. The length of JCHAR string data.
encryptkey	Input. JCHAR string encrypt key. If *encryptkey* is NULL, has a length of zero, or just a single blank space, this API will fail.
keylen	Input. The length of the JCHAR encrypt key string.
type	Input. The type of encryption algorithm.

Returns

This API returns these values:

Value	Description
1	JDEENCRYTT_SUCCESS. Indicates the API was successful.
0	JDEENCRYPT_FAILURE. Indicates that the API failed.

Example

```
/* Declare variables associated with jdeEncryptWKey */
* declare example data
 *
 *******************************************/
 JCHAR   EKey[32]=_J("12345678");
 int  EKeyLen=8;
 JCHAR   EData[1024]=_J("TESTDATA");
 int  EDataLen=8;
 BYTE   EncryptedData[1024]={0};
 int  EncryptedLen;
 int   type=eEVPTripleDES;
  int   iRet;

  iRet=jdeEncryptWKey(EncryptedData,&EncryptedLen,EData,EDataLen,EKey,EKey⇒
Len,type);
  if (iRet!= JDEENCRYPT_SUCCESS)
  {
  jdePrintf("jdeEncryptWKey failed\n");
  } /* END IF */
```


jdeDecryptWKey

Syntax

```
int jdeDecryptWKey(JCHAR *outbuf,
     int *poutlen,
     BYTE *inbuf,
     int inlen,
     JCHAR *encryptkey,
     int keylen,
     int type
     );
```

Description

jdeDecryptWKey performs decryption with input encrypted data bytes and JCHAR encrypt key string, and then return decrypted JCHAR string with its length. The TripleDES algorithm is used.

Parameters

Parameter	Description
outbuf	Output. Decrypted JCHAR string. If null, the API fails.
poutlen	Output. Length of decrypted JCHAR string.
inbuf	Input. Encrypted bytes. If null or the length is zero, the API fails.
inlen	Input. Length of encrypted bytes.
encryptkey	Input. JCHAR string encrypt key. Minimum length is 4; maximum length is 16. If the length is zero or it contains only a blank space, the API fails.
keylen	Input. Length of JCHAR encrypt key string.
type	Input. Type of encryption algorithm.

Returns

This API can return these values:

Value	Description
1	JDEENCRYPT_SUCCESS. Indicates that the API succeeded.
0	JDEENCRYPT_FAILURE. Indicates that the API failed.

Example

```
/* Declare variables associated with jdeEncryptWKey */
* declare example data
 *
 *******************************************/
 JCHAR   EKey[32]=_J("12345678");
 int   EKeyLen=8;
 JCHAR   EData[1024]=_J("TESTDATA");
```

```
 int   EDataLen=8;
 BYTE   EncryptedData[1024]={0};
 int   EncryptedLen;
 int   type=eEVPTripleDES;
  int   iRet;
 JCHAR   DecryptedData[1024]={0};
 int  DecryptedLen;

  iRet=jdeEncryptWKey(EncryptedData,&EncryptedLen,EData,EDataLen,EKey,EKey⇒
Len,type);
  if (iRet!= JDEENCRYPT_SUCCESS)
  {
  jdePrintf("jdeEncryptWKey failed\n");
  } /* END IF */


  iRet=jdeDecryptWKey(DecryptedData,&DecryptedLen,EncryptedData,Encrypted⇒
Len,EKey,EKeyLen,type);
  if (iRet!= JDEENCRYPT_SUCCESS)
  {
 jdePrintf("jdeDecryptWKey failed\n");
  } /* END IF */
```

JDB_TextSearchClearSelection

Syntax

`JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchClearSelection(HREQUEST hRequest);`

Description

This API clears all select criteria for text searches.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context in which the text search selection criteria are to be cleared.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The selection criteria we cleared successfully.
JDEDB_FAILED	The selection criteria were not cleared.

Example

```
JDBDB_RESULT nResult;

nResult = JDB_TextSearchClearSelection(hRequest);
```

JDB_TextSearchClearSequencing

Syntax

JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchClearSequencing(HREQUEST *hRequest*0;

Description

This API clears all sequencing criteria for text searches.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context in which the text search sequencing criteria are to be cleared.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The sequencing criteria we cleared successfully.
JDEDB_FAILED	The sequencing criteria were not cleared.

Example

```
JDBDB_RESULT nResult;

nResult = JDB_TextSearchClearSequencing(hRequest);
```

JDB_TextSearchCloseView

Syntax

JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchCloseView(HREQUEST *hRequest*);

Description

This API closes a business view that was opened for text searches.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context in which the business view is to be closed.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The business view was closed successfully.
JDEDB_FAILED	The business view was not closed.

Example

```
DBDB_RESULT nResult;

nResult = JDB_TextSearchCloseView(hRequest);
```

JDB_TextSearchFetch

Syntax

```
JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchFetch(
      HREQUEST hRequest,
      void FAR * lpValue,
      void FAR * score,
      void FAR * summary,
      int nLock
      );
```

Description

This API fetches a row of text search results.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context in which the text search was issued.
lpValue	Output, required. The business view data structure where the text search results are stored.
summary	Output, required. The summary that describes the context of the match in the text search results.
nLock	Input, required. Not used.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The text search results were fetched successfully.
JDEDB_FAILED	The search results were not fetched.

Example

```
DSV0101C value; /* User-defined struct */
MATH_NUMERIC mnScore;
JCHAR szSummary[256] = { 0 };
JDBDB_RESULT nResult;

nResult = JDB_TextSearchFetch(hRequest, &value, &mnScore, szSummary, 0);
```

JDB_TextSearchOpenView

Syntax

```
JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchOpenView(
      HUSER hUser,
      NID szBob,
      JCHAR * lpszDataSource,
      HREQUEST * hRequest
      );
```

Description

This API opens a business view for text searches. You must call this API before issuing any text searches.

Parameters

Parameter	Description
hUser	Input, required. The user handle that defines the context in which to open the business view.
szBob	Input, required. The business view that defines the data and media objects to include in the text search index.
lpszDataSource	Input, required. The data source that defines the location of the data and media objects to search.
hRequest	Output, required. The request handle to use for subsequent text search operations.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The business view was opened successfully.
JDEDB_FAILED	The business view was not opened.

Example

```
NID szBob = _J("V0101C");
JCHAR * lpszDataSource = _J("My Business Data");
HREQUEST hRequest = NULL;
JDBDB_RESULT nResult;

nResult = JDB_TextSearchOpenView(hUser, szBob, lpszDataSource, &hRequest);
```

JDB_TextSearchSelect

Syntax

```
JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchSelect(
       HREQUEST hRequest,
       const JCHAR * lpKeyWordsValue,
       JDB_TEXTSEARCH_VALUE_TYPE nValueType,
       JDB_TEXTSEARCH_KEYWORD_OPTION nKeyOption
       );
```

Description

This API issues a text search. You must open a business view with **JDB_TextSearchOpenView** before you use this API to issue a text search.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context in which to issue the text search.
lpKeyWordsValue	Input, required. The keywords or Verity Query Language (VQL) string for the text search.
nValueType	Input, required.`JDB_TEXTSEARCH_VALUE_TYPE_KEYWORDS` = *lpKeyWordsValue* is a keywords string.`JDB_TEXTSEARCH_VALUE_TYPE_VQL` = *lpKeyWordsValue* is a VQL string.
nKeyOption	Input, required. A bitmap that indicates how to treat keywords. You can use either or both of these values:`JDB_TEXTSEARCH_KEYWORD_OPT_CASE_SENSITIVE` = match text using case sensitivity.`JDB_TEXTSEARCH_KEYWORD_OPT_SIMILARITY` = match similar words.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The text search was issued successfully.
JDEDB_FAILED	The text search was not issued.

Example

```
JDBDB_RESULT nResult;

nResult = JDB_TextSearchSelect(hRequest, _J("financial company"), JDB_TEXTSEARCH_⇒
VALUE_TYPE_KEYWORDS, JDB_TEXTSEARCH_KEYWORD_OPT_SIMILARITY);
```

JDB_TextSearchSetSelection

Syntax

```
JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchSetSelection(
       HREQUEST hRequest,
       LPNEWSELECT lpSelect,
       ushort nNum,
       JDEDB_SET nSet
       );
```

Description

This API sets the selection criteria for a text search.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context to which the text search selection criteria is to be applied.
lpSelect	Input, required. The text search selection criteria.
nNum	Input, required. The number of selection fields.
nSet	Input, required.JDEDB_SET_REPLACE = replace the existing selection criteria.JDEDB_SET_APPEND = append to the existing selection criteria.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The text search selection criteria were set successfully.
JDEDB_FAILED	The text search selection criteria were not set.

Example

```
NEWSELECTSTRUCT lpSelect[1];
JCHAR szTemp[10];
MATH_NUMERIC mnTemp;
JDBDB_RESULT nResult;

/* Populate the select structure. */
jdeNIDcpy(lpSelect[0].Item1.szDict, NID_AN8);
jdeNIDcpy(lpSelect[0].Item1.szTable, NID_F0101);
lpSelect[0].Item1.idInstance = 0;
lpSelect[0].nValues = 1;
lpSelect[0].nCmp = JDEDB_CMP_EQ;
lpSelect[0].nAndOr = JDEDB_ANDOR_AND;
jdeSprintf(szTemp, _J("%d"), 4100);
ParseNumericString(&mnTemp, szTemp);
lpSelect[0].lpValue = &mnTemp;
lpSelect[0].nParen = JDEDB_PAREN_NONE;

nResult = JDB_TextSearchSetSelection(hRequest,
 lpSelect, 1, JDEDB_SET_REPLACE);

/* Search in Media Objects that are associated with a GT Type Name. */
NEWSELECTSTRUCT lpSelect[1];
JCHAR szTemp[10];
MATH_NUMERIC mnTemp;
JDBDB_RESULT nResult;

jdeNIDcpy(lpSelect[0].Item1.szDict, _J("GT4801A"));
/* Provides a GT type name that searching media objects associate with here */
jdeNIDcpy(lpSelect[0].Item1.szTable, JDB_MEDIAOBJECT);
/* JDB_MEDIAOBJECT is a constant to indicate the usage of a GT type name in the⇒
 above line */
lpSelect[0].Item1.idInstance = 0;
lpSelect[0].nValues = 1;
lpSelect[0].nCmp = JDEDB_CMP_EQ;
lpSelect[0].nAndOr = JDEDB_ANDOR_AND;
jdeSprintf(szTemp, _J("%d"), 4100);
ParseNumericString(&mnTemp, szTemp);
lpSelect[0].lpValue = &mnTemp;
lpSelect[0].nParen = JDEDB_PAREN_NONE;

nResult = JDB_TextSearchSetSelection(hRequest,
 lpSelect, 1, JDEDB_SET_REPLACE);
```

JDB_TextSearchSetSequencing

Syntax

```
JDERTN (JDEDB_RESULT) JDEWINAPI JDB_TextSearchSetSequencing(
```



```
HREQUEST hRequest,
LPSORT lpSSort,
ushort nNum,
JDEDB_SET nSet
);
```

Description

This API sets the sequencing criteria for a text search.

Parameters

Parameter	Description
hRequest	Input, required. The request handle that defines the context to which the text search selection criteria is to be applied.
lpSort	Input, required. The text search sequencing criteria.
nNum	Input, required. The number of sort fields.
nSet	Input, required.`JDEDB_SET_REPLACE` = replace the existing sequencing criteria.`JDEDB_SET_APPEND` = append to the existing sequencing criteria.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	The text search sequencing criteria were set successfully.
JDEDB_FAILED	The text search sequencing criteria were not set.

Example

```
SORTSTRUCT lpSort[2];
JDBDB_RESULT nResult;

/* Populate the sort structure. */
jdeNIDcpy(lpSort[0].Item.szDict, NID_AN8);
lpSort[0].Item.idInstance = 0;
jdeNIDcpy(lpSort[0].Item.szTable, NID_F0101);
lpSort[0].nSort = JDEDB_SORT_ASC;

jdeNIDcpy(lpSort[1].Item.szDict, NID_ALPH);
lpSort[1].Item.idInstance = 0;
jdeNIDcpy(lpSort[1].Item.szTable, NID_F0101);
lpSort[1].nSort = JDEDB_SORT_ASC;

nResult = JDB_TextSearchSetSequencing(hRequest, lpSort, 2, JDEDB_SET_REPLACE);
```


TextSearchFullIndexing

Syntax

```
JDERTN (TEXTSEARCH_INDEXING_RESULT) JDEWINAPI TextSearchFullIndexing(
          HUSER lpUser,
          JCHAR* lpszDataSource,
          JCHAR* lpszBusinessView
          );
```

Description

This API builds a text search index which is an indexed copy of JD Edwards EnterpriseOne data and media objects that is optimized for text searches.

Parameters

Parameter	Description
lpUser	Input, required. The user handle that defines the context in which to build the text search index.
lpszDataSource	Input, required. The data source that defines the location of the data to include in the text search index.
lpszBusinessView	Input, required. The business view that defines the data and media objects to include in the text search index.

Additional Notes

The full text search index build leaves the existing index intact (if one exists) for the duration of the build process. Doing so ensures that searches remain functional throughout the course of the build. When the text search index build is complete, this API clears the previous text search index build instance.

Returns

This API can return these values:

Value	Description
TEXTSEARCH_INDEXING_SUCCESS	The full text search index build completed successfully.
TEXTSEARCH_INDEXING_FAIL	The full text search index build failed.

Example

```
TEXTSEARCH_INDEXING_RESULT nResult;
nResult = TextSearchFullIndexing(lpUser, _J("My Business Data Source"), _J⇒
("V0101C"));
```

TextSearchIncrementIndexing

Syntax

```
JDERTN (TEXTSEARCH_INDEXING_RESULT) JDEWINAPI TextSearchIncrementIndexing (
          HUSER lpUser,
          JCHAR* lpszBusinessView,
          JCHAR* lpszDataSourceOverride,
          TEXTSEARCH_INDEX_MODE mode,
          LPKEYINFO lpKeyInfo,
          short nKeys
          );
```

Description

This API performs an incremental text search index build which modifies an existing text search index to reflect a change to a single row of JD Edwards EnterpriseOne data or attached media objects.

Parameters

Parameter	Description
lpUser	Input, required. The user handle that defines the context in which to modify the text search index.
lpszBusinessView	Input, required. The business view that defines the data and media objects to include in the text search index.
lpszDataSourceOverride	Input, optional. The data source that defines the location of the data to include in the text search index. If this is not provided, the API uses the OCM to determine the data source.
mode	Input, required. The operation to perform: TEXTSEARCH_INDEX_INSERT, TEXTSEARCH_INDEX_UPDATE, or TEXTSEARCH_INDEX_DELETE.
lpKeyInfo	Input, required. The primary key data that defines the row to modify in the text search index.
nKeys	Input, required. The number of keys defined in *lpKeyInfo*.

Additional Notes

This API provides a mechanism to keep text search indexes synchronized with changes to the corresponding JD Edwards EnterpriseOne data and media objects. However, incremental builds reduce the efficiency of text searches. It is recommended that administrators schedule regular full text search index builds or optimizations in conjunction with applications that call this API.

Returns

This API can return these values:

Value	Description
TEXTSEARCH_INDEXING_SUCCESS	The full text search index build completed successfully.
TEXTSEARCH_INDEXING_FAIL	The full text search index build failed.

Example

```
LPKEYINFO lpKeyInfo;
JCHAR szTemp[10];
MATH_NUMERIC mnTemp;
TEXTSEARCH_INDEXING_RESULT nResult;

/* Populate the key information. */
jdeSprintf(szTemp, _J("%d"), 4100);
ParseNumericString(&mnTemp, szTemp);

lpKeyInfo = (LPKEYINFO)jdeAlloc(COMM_POOL, sizeof(KEYINFO), MEM_ZEROINIT);
jdeNIDcpy(lpKeyInfo[0].szDict, NID_AN8);
jdeNIDcpy(lpKeyInfo[0].szTable, NID_F0101);
lpKeyInfo[0].idInstance = 0;
lpKeyInfo[0].lpJDEValue = &mnTemp;

nResult = TextSearchIncrementIndexing(lpUser, _J("V0101C"), NULL, TEXTSEARCH_INDEX_
INSERT, lpKeyInfo, (short)1);
```

TextSearchIndexClearing

Syntax

```
JDERTN (TEXTSEARCH_INDEXING_RESULT) JDEWINAPI TextSearchIndexClearing (
          HUSER lpUser,
          JCHAR* lpszDataSourceName,
          JCHAR* lpszBusinessView
          );
```

Description

This API clears a text search index build. Text searches do not work on a text search index that has been cleared.

Parameters

Parameter	Description
lpUser	Input, required. The user handle that defines the context in which to modify the text search index.
lpszDataSource	Input, required. The data source that defines the location of the data included in the text search index.
lpszBusinessView	Input, required. The business view that defines the data and media objects to include in the text search index.

Additional Notes

Clear a text search index when users no longer need to issue text searches against it. Clearing a text search index releases the system resources associated with it.

Returns

This API can return these values:

Value	Description
TEXTSEARCH_INDEXING_ SUCCESS	The full text search index build completed successfully.
TEXTSEARCH_INDEXING_FAIL	The full text search index build failed.

Example

```
TEXTSEARCH_INDEXING_RESULT nResult;

nResult = TextSearchIndexClearing(lpUser, _J("My Business Data Source"), ⇒
_J("V0101C"));
```

TextSearchIndexOptimizing

Syntax

```
JDERTN (TEXTSEARCH_INDEXING_RESULT) JDEWINAPI TextSearchIndexOptimizing (
          HUSER lpUser,
          JCHAR* lpszDataSourceName,
          JCHAR* lpszBusinessView
          );
```

Description

This API optimizes a text search index build which reorganizes an existing text search index to facilitate faster text searches.

Parameters

Parameter	Description
lpUser	Input, required. The user handle that defines the context in which to optimize the text search index.
lpszDataSource	Input, required. The data source that defines the location of the data included in the text search index.
lpszBusinessView	Input, required. The business view that defines the data and media objects included in the text search index.

Additional Notes

Incremental text search index builds reduce the efficiency of text searches. Optimizing text search index reorganizes the data modified by incremental builds in order to improve text search performance.

Returns

This API can return these values:

Value	Description
TEXTSEARCH_INDEXING_ SUCCESS	The full text search index build completed successfully.
TEXTSEARCH_INDEXING_FAIL	The full text search index build failed.

Example

```
TEXTSEARCH_INDEXING_RESULT nResult;

nResult = TextSearchIndexOptimizing(lpUser, _J("My Business Data Source"), _J⇒
("V0101C"));
```

Dynamic Logging APIs

This section discusses dynamic logging APIs.

You use these APIs to call the dynamic logging functions. The APIs insert the filename and line number info into the log message that is output when the logging level changes. These logging levels affect what is written to the jdedebug.log files. The logging APIs will write messages to the jdedebug.log files to indicate the logging levels have changed

If you set any dynamic logging options using these APIs, those settings take precedence over the settings in the jde.ini file, but only for kernels that have been changed with these APIs. All other kernels use the jde.ini settings.

chgOutputLoggingLevel

Syntax

```
bool chgOutputLoggingLevel (int level);
```

Description

Use this API to turn debug logging on or off. If output logging is turned off, all other logging levels are also turned off, except **EVN SaveEVNDoc** and **IEO Trace** logging levels. Those settings are not affected by **chgOutputLoggingLevel**, which you must disable manually if necessary.

If output logging is turned off, calling this API to turn it off again does not result in another log message indicating that output logging was turned off.

If the level parameter is something other than 0 or 1, an error message is written to the log indicating that the API was called with incorrect parameters. This message is not written if output logging is turned off. Output logging must be turned on for any other logging levels to have an effect. If output logging is turned off, but other logging levels are turned on, the logging output is formatted and prepared for output, but is not written to the log file. This results in a performance reduction without the benefit of extra logging.

Ini File Equivalent

```
[DEBUG]
Output=FILE
LogErrors=1
```

Parameters

Parameter	Description
level	Input, required. The logging level. Must be either 0 or 1.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turns on output logging and writes a message in the log to indicate that it was⇒
 turned on */

chgOutputLoggingLevel(0);
/* turns off debug logging, and turns off all other logging levels except for IEO⇒
 Trace and EVN Save Doc levels. These must be turned off manually if appropriate⇒
 */
```

chgIPCTraceLevel

Syntax

```
bool chgIPCTraceLevel (int level);
```

Description

Use this API to change the IPC Trace logging level.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

Ini File Equivalent

```
[JDEIPC]
ipcTrace=x
```

Parameters

Parameter	Description
level	Input, required. The logging level. Must be in the range 0–3.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turn on output logging so log messages will be seen */
chgIPCTraceLevel(2);
/* set IPC Trace to level 2 */

chgIPCTraceLevel(0); /* turn off IPCTrace */
chgOutputLogging(0); /* turn off debug logging (and all other logging types) */
```

See Also

chgOutputLoggingLevel(1)

chgNetTraceLevel

Syntax

```
bool chgNetTraceLevel (int level);
```

Description

Use this API to change the Net Trace logging level.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

Ini File Equivalent

```
[JDENET]
netTrace=x
```

Parameters

Parameter	Description
level	Input, required. The logging level. Must be in the range 0–7.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1); /* turn on output logging so log messages will be seen⇒
 */
chgNetTraceLevel (2); /* set Net Trace to level 2 */

chgNetTraceLevel (0); /* turn off Net Trace */
chgOutputLoggingLevel(0); /* turn off output logging */
```

See Also

chgOutputLoggingLevel(1)

chgTAMTraceLevel

Syntax

```
bool chgTAMTraceLevel (int level);
```

Description

Use this API to change the TAM Trace logging level.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

Ini File Equivalent

```
[DEBUG]
TAMTraceLevel=x
```

Parameters

Parameter	Description
level	Input, required. The logging level. Must be in the range 0–9.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1); /* turn on output logging so log messages will be seen⇒
 */
chgTAMTraceLevel (5); /* set TAM Trace to level 5 */

chgTAMTraceLevel (0); /* turn off TAM trace logging */
chgOutputLoggingLevel(0); /* turn off debug logging */
```

chgCMTraceLevel

Syntax

```
bool chgCMTraceLevel (int level);
```

Description

Use this API to change the CM Trace logging level.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

chgCMTraceLevel works in conjunction with **chgCMTraceFilter**. You must set the cache logging filter with this API. If the filter is not set when you call **chgCMTraceLevel**, the cache logging filter will be set to “ALL”, which means that all cache logging messages will be seen, assuming that output logging is also on.

Ini File Equivalent

```
[DEBUG]
CMTrace=1
CMTraceFilter=ALL
```

Parameters

Parameter	Description
level	Input, required. The logging level. Must be either 0 or 1.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turn on output logging so log messages will be seen */
chgCMTraceFilter(_J("ALL"));
/* set filter to ALL meaning all cache logging will be seen*/
chgCMTraceLevel (1);
/* turn cache logging on */

chgCMTraceLevel (0);
/* turn cache logging off */
chgOutputLoggingLevel(0);
/* turn off debug logging */
```


chgCMTraceFilter

Syntax

```
bool chgCMTraceFilter (jchar filter);
```

Description

Use this function to set the cache logging filter. This filter enables or prevents certain cache messages from being seen.

Ini File Equivalent

```
[DEBUG]
CMTraceFilter=nnnnnnnn
```

Parameters

Parameter	Description
filter	Input, optional. A comma or semicolon-delimited string of up to 256 characters. If the filter is longer than 256 characters, it is truncated at the 256th character. If the parameter is blank or consists of white space, the filter is set to “ALL”, meaning that all cache logging should be output to the log.

Returns

This API always returns *true*.

Example

```
chgOutputLoggingLevel(1);
/* turn on output logging so log messages will be seen */
chgCMTraceFilter(_J("ALL"));
/* set filter to ALL meaning all cache logging will be seen*/
chgCMTraceLevel (1);
/* turn cache logging on */
```

chgPSThreadTraceLevel

Syntax

```
bool chgPSThreadTraceLevel (int level);
```

Description

Use this API to change the PSThread Trace logging level.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

Note. Setting this value to 3 results in an extremely large amount of logging and will severely affect performance on the Enterprise server. Do not use level 3 unless you are working with PSThread code and understand the performance degradation.

Ini File Equivalent

```
[DEBUG]
ThreadTraceLevel=x
```

Parameters

Parameter	Description
level	Input, required. The trace level. Must be in the range 0–3.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turn on output logging so log messages will be seen */
chgPSThreadTraceLevel (1);
/* set PSThread Trace to level 1 */

chgPSThreadTraceLevel (0);
/* turn off PSThread logging */
chgOutputLoggingLevel(0);
/* turn off debug logging */
```

chgSecTraceLevel

Syntax

```
bool chgSecTraceLevel (int level);
```

Description

Use this API to change the security trace logging level.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

Ini File Equivalent

```
[SECURITY]
secTrace=x
```

Parameters

Parameter	Description
level	Input, required. The trace level. Must be in the range 0–2.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turn on output logging so log messages will be seen */
chgSecTraceLevel (2);
/* set security Trace to level 2 */

chgSecTraceLevel (0);
/* turn off secTrace */
chgOutputLoggingLevel(0);
/* turn off debug logging */
```

chgSaveEVNDoc

Syntax

```
bool chgSaveEVNDoc (int level);
```

Description

Use this API to change the Save EVN Doc value.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

This function is available only with EVN kernels.

Ini File Equivalent

```
[Interoperability]
SaveEVNDoc=1
```

Parameters

Parameter	Description
level	Input, required. The save value. Must be either 0 or 1.

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turn on output logging so log messages will be seen */
chgSaveEVNDoc (1);
/* set Save EVN Doc level to 1 */

chgSaveEVNDoc (0);
/* turn off SaveEVNDoc*/
```

chgIEOTraceLevels

Syntax

```
bool chgIEOTraceLevels (int level);
```

Description

Use this API to change the IEO Trace levels.

If you call this function with a valid non-zero level parameter, you should also ensure that output logging is on. If not, no log messages will be written and performance will suffer. Turn output logging on by calling: **chgOutputLoggingLevel***(1);*.

If you call this function with a level parameter outside the valid range, an error message is written to the logs. This message does not appear if output logging is turned off.

Ini File Equivalent

```
[Interoperability]
Level=PERF,TRACE,DOC,DATA,EVENTS
```

Parameters

The parameter *level* is a bitmap of the following values. To turn on the levels, add together the value in the table below for the values you want to turn on. All other levels will be turned off.

Logging Type	Value	Hex
EVENTS	1	0x01
DATA	2	0x02
PERF	4	0x04
DOC	8	0x08
TRACE	16	0x10

Returns

This API can return these values:

Value	Description
True	Indicates that the API turned logging on.
False	Indicates that the API turned logging off.

Example

```
chgOutputLoggingLevel(1);
/* turns on output logging and outputs a message in the log to indicate that it⇒
 was turned on */

chgOutputLoggingLevel(0);
/* turns off debug logging, and turns off all other logging levels except for IEO⇒
 Trace and EVN Save Doc levels. These must be turned off manually if appropriate */
```

jdeCache APIs

jdeCacheAdd

Syntax

```
JDECM_RESULT jdeCacheAdd (HCACHE hCache, void * lpData, long nDataSize)
```

Description

jdeCacheAdd adds a record to the cache identified by the cache handle *hCache* that was obtained from the call to **jdeCacheInit**. The sequence order of the added record is determined by the index that was passed to **jdeCacheInit**.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
lpData	Input. This is the full structure from which the offsets of the columns that make up the index keys were established. In a JDB environment, the structure represents one record. Use **memset()** on this structure before adding elements to it.
nDataSize	Input. The size of the structure record that is being passed with the data.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];
memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");
/*set up index structure*/
...
/*initialize cache*/
...
/*add a record*/
memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing  12345");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");
/*do something*/
```



```
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
      printf("jdeCacheTerminate Failed \n");
```

jdeCacheClear

Syntax

```
JDECM_RESULT jdeCacheClear(HCACHE hCache);
```

Description

jdeCacheClear deletes all the records in the cache. It does not terminate the cache.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];
memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");
/*set up index structure*/
...
/*initialize cache*/
...
 /*add some records*/
...
/*do something*/
...
if (jdeCacheClear(hCache)==JDECM_FAILED)
      printf("jdeCacheClearCache Failed !\n");
if (jdeCacheGetNumRecords(hCache)!=0)
      printf("Still have records in cache after jdeCacheClearCache !\n");
/*do something*/
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
      printf("jdeCacheTerminate Failed \n");
```

jdeCacheCloseCursor

Syntax

```
JDECM_RESULT jdeCacheCloseCursor (HCACHE hCache, HJDECURSOR hCursor);
```

Description

jdeCacheCloseCursor closes a cursor pointing to the data set within a cache. The cursor is closed regardless of its position within the data set. A subsequent call to any cursor API except for **jdeCacheOpenCursor** will fail.

Note. For multiple opened HJDECURSORs, this API must be called to close all opened HJDECURSOR before the cache is terminated.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache, associated with *hcache*, that you want to close. Use this cursor handle in calls to Data Manipulation APIs to reference records in the cache.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");
/*set up index structure*/
...
/*initialize cache*/
...
/*add some records*/
...
/*open a cursor*/
...
/*do something*/
...
if (jdeCacheCloseCursor(hCache,hCursor)==JDECM_FAILED)
      printf("jdeCacheCloseCursor Failed\n");
/*do something*/
...
```

```
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheDelete

Syntax

```
JDECM_RESULT jdeCacheDelete (HCACHE hCache, HJDECURSOR hCursor)
```

Description

jdeCacheDelete deletes the record that is pointed to by the cursor. If the cache is not empty and the cursor is open, the record currently being pointed to by HJDECURSOR is deleted.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hcache*. Use this cursor handle in calls to Data Manipulation APIs to reference records in the cache.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];
char ret_Data[100];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 10;
Index[0].CacheKey[1].nSize=10;
```

```
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
..
/*add  records*/
memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing");
strcpy(Data+10, "12345");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");

memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing");
strcpy(Data+10, "67890");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");

/*open a cursor*/
...
/*fetch position*/
memset(ret_Data,0x00,sizeof(ret_Data));
if (jdeCacheFetchPosition(hCache,hCursor,Data,2,ret_Data,sizeof(ret_Data))==JDECM_⇒
FAILED)
     printf("Record not found\n");

/*delete the key that the cursor points at*/
if (jdeCacheDelete(hCache,hCursor)==JDECM_FAILED)
     printf("Delete record failed\n");
...
/*do something*/
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheDeleteAll

Syntax

```
JDECM_RESULT jdeCacheDeleteAll ⇒
(HCACHE hCache, HJDECURSOR hCursor, void * lpKey, short int nNumKeys)
```

Description

jdeCacheDeleteAll deletes all records matching the given key from the cache identified by the cache handle *hCache* that was obtained from the call to **jdeCacheInit**. This API works well when deleting records based on a partial key.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hcache*. Use this cursor handle in calls to Data Manipulation APIs to reference records in the cache.
lpKey	Input. Use **memset()** on this key before adding elements to it. The number, order, and type of the elements in this structure must match that of the defined index keys.
nNumKeys	Input. The number of keys in *lpKey*. If this number is less than the total defined in **jdeCacheInit**, the key is treated as a partial key.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 10;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
...

/*add  records*/
memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing");
strcpy(Data+10, "12345");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
```



```
            printf("Add record failed !!\n");


    memset(Data,0x00,sizeof(Data));
    strcpy(Data,"Testing");
    strcpy(Data+10, "67890");
    if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
            printf("Add record failed !!\n");

    /*open a cursor*/
    ...

    /*delete all the records with the first key is "Testing"*/
    if (jdeCacheDeleteAll(hCache,hCursor,Data,1)==JDECM_FAILED)
         printf("Delete record failed\n");
    ...

    /*do something*/
    ...

    if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
         printf("jdeCacheTerminate Failed \n");
```

jdeCacheFetch

Syntax

```
JDECM_RESULT jdeCacheFetch ⇒
(HCACHE hCache, HJDECURSOR hCursor, void *lpData, void * lpNull);
```

Description

JdeCacheFetch advances the cursor to the next record in the dataset relative to the current cursor position and retrieves the record. This becomes the new position associated with HJDECURSOR. This API does not search for a record in the cache. Instead, it fetches the next record after the record that the passed HJDECURSOR is currently pointing to. The API is ideal for performing sequential fetches.

First, the passed HJDECURSOR is checked to see if it's the current ACTIVE cursor. If it is not, the Cursor Manager is referenced and the appropriate HJDECURSOR is obtained and made to be the active HJDECURSOR. If the cache is not empty and the cursor is open, the cursor is advanced to the next record in the data set related to the current cursor position before fetching the record at the new cursor position.

The *IpData* must be at least the same size as the data set that was added. If you have added different sizes of data sets to the same cache, you must use the size of the largest data set in *IpData*. If you do not, you might cause memory to be overridden.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hcache*. Use this cursor handle in calls to Data Manipulation APIs to reference records in the cache.
lpData	Output. Pointer structure to be filled with the fetched data.
lpNull	Input. Must be NULL. Reserved for future use.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];
char ret_Data[100];
void lpNull= NULL;
memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 10;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
...

/*add  records*/
memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing");
strcpy(Data+10, "12345");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
```



```
          printf("Add record failed !!\n");
    memset(Data,0x00,sizeof(Data));
    strcpy(Data,"Testing");
    strcpy(Data+10, "67890");
    if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
          printf("Add record failed !!\n");

    /*open a cursor*/
    ...

    /*fetch position the cursor to the first record that has the key "Testing"*/
    memset(ret_Data,0x00,sizeof(ret_Data));
    if (jdeCacheFetchPosition(hCache,hCursor,Data,1,ret_Data,sizeof(ret_Data))==JDECM_⇒
    FAILED)
         printf("Record not found\n");

    /*continue fetching*/
    memset(ret_Data,0x00,sizeof(ret_Data));
    if (jdeCacheFetch(hCache,hCursor,Data,lpNull)==JDECM_FAILED)
         printf("jdeCacheFetch Failed\n");
    ...

    /*do something*/
    ...
    if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
         printf("jdeCacheTerminate Failed \n");
```

jdeCacheFetchPosition

Syntax

```
JDECM_RESULT jdeCacheFetchPosition ⇒
(HCACHE hCache, HJDECURSOR hCursor, void *lpKey, short int nNumColsInKey, ⇒
void * lpData, long nDataSize)
```

Description

jdeCacheFetchPosition retrieves the record that corresponds to the given key. You can use this API to position the HJDECURSOR at a specific location within the data set. When you make a call to **jdeCacheFetch**, it starts fetching from the next position. This is useful for making sequential fetches beginning at a specific location in the cache.

First, the passed HJDECURSOR is checked to see if it's the current active cursor. If it is not, then the Cursor Manager is accessed and the appropriate HJDECURSOR is obtained and made to be the active HJDECURSOR. If the cache is not empty and the cursor is open, then the passed number of keys is checked to see if they constitute a partial index. If they do, then the partial key is used to fetch the first record that matches the given partial key. If the cache is not empty and the cursor is open, the record that matches the given key will be returned and the cursor will point to the founded record.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hcache*. Use this cursor handle in calls to Data Manipulation APIs to reference records in the cache.
lpKey	Input. Use **memset()** on this structure before adding elements to it. The number, order, and type of the elements in this structure must match that of the defined index keys.
nNumColsInKey	Input. The number of columns in the passed key with which you want the search to be made. This allows for partial indexing. Only the columns starting from the first column to the number of columns specified may be used. They must be in succession. If this number is less than the total that was defined in **jdeCacheInit**, then the key is treated as a partial key.
lpData	Output. The offsets of the columns constituting the index keys that were established from this full structure. In a JDB environment, the structure represents one record.
nDataSize	Input. Use **sizeof()** to get the size of the structure that is being passed with the data.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];
char ret_Data[100];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 10;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;
```



```
/*initialize cache*/
...

/*add  records*/
memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing");
strcpy(Data+10, "12345");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");

memset(Data,0x00,sizeof(Data));
strcpy(Data,"Testing");
strcpy(Data+10, "67890");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");

/*open a cursor*/
...

/*fetch position the cursor to the first record that has the key "Testing  ⇒
 67890"*/
memset(ret_Data,0x00,sizeof(ret_Data));
if (jdeCacheFetchPosition(hCache,hCursor,Data,2,ret_Data,sizeof(ret_Data))==JDECM_⇒
FAILED)
     printf("Record not found\n");
...

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheFetchPositionByRef

Syntax

`JDECM_RESULT jdeCacheFetchPositionByRef(HCACHE hCache, void **lplpData)`

Description

jdeCacheFetchPositionByRef retrieves the memory address of the one and only one data block in cache.

If more than one record exists in cache, this API will return failed. If the cache is empty, this API will return failed. This API will always return the same memory address.

Note. Do not free the returned data pointer address. This action may cause a memory violation problem on certain platforms.

Parameters

Parameter	Description
hCache	Pointer to Data Cache memory. The cache handle must be initialized before calling this API. If the cache handle is NULL, this API will return fail. If the cache handle is not initialized or is an invalid memory, it will cause a memory violation problem. Therefore, the best way to avoid the memory violation problem is to always set the cache handle to NULL where it is defined. Input
lplpData	Output. Pointer to the data block in cache.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
void *lpData=NULL; /*a data pointer*/

/*Initialize cache handle and adds a record*/

...
if (jdeCacheFetchPositionByRef(hCache,&lpData)==JDECM_FAILED)
      printf("jdeCacheFetchPositionByRef Failed !! \n");
...

/*Free the cache*/

...
```

jdeCacheGetIndex

Syntax

```
long int jdeCacheGetIndex(HCACHE hCache, HJDECURSOR hCursor)
```

Description

jdeCacheGetIndex enables you to retrieve the ID of the index to which the cursor is pointing. Use this API when you are using multiple indexes.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hcache*. Use this cursor handle used in calls to Data Manipulation APIs to reference records in the cache.

Returns

This API can return this value:

long int The ID of the Index to which the cursor is pointing.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[2];
short int nNumindexes   =  2;
char szCacheName[20];
long int nKeyID = 2;

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up multiple indexes structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT)*2);
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

Index[1].nKeyID = 2;
Index[1].nNumSegments = 1;
Index[1].CacheKey[0].nOffset = 10;
Index[1].CacheKey[0].nSize=5;
Index[1].CacheKey[0].idDataType = EVDT_STRING;

/*initialize multiple indexes*/
if (jdeCacheInitMultipleIndex(hUser, &hCache, szCacheName, Index, nNumindexes)==⇒
JDECM_FAILED)
{
printf("Initialize cache failed \n");
      /*do something*/
      return ;
```



```
}

/*add some records*/
...

/*open cursor*/
...
if (jdeCacheSetIndexNumCursors(hCache,hCursor,nKeyID)==JDECM_FAILED)
      printf("Set Index ID failed !!\n");

/*do something*
if (jdeCacheGetIndexNumCursors(hCache,hCursor)!=nKeyID)
      printf("Cursor point at the wrong Index!!\n");

 /*do something*/
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheGetNumCursors

Syntax

```
short int jdeCacheGetNumCursors(HCACHE hCache)
```

Description

jdeCacheGetNumCursors returns the total number of cursors currently opened in the cache identified by *hCache*.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.

Returns

This API can return this value:

short int Total number of cursors currently in the cache.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");
```



```
/*set up an index structure*/
...

/*initialize cache*/
...

/*add some records*/
...

/*open cursor*/
...

if (jdeCacheGetNumCursors(hCache)<1)

      printf("No cursor was opened in cache[%s]\n",szCacheName);

 /*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheGetNumRecords

Syntax

```
long int jdeCacheGetNumRecords (HCACHE hCache);
```

Description

jdeCacheGetNumRecords returns the total number of records currently stored in the cache identified by *hCache*. You can use it to check if the cache is empty.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.

Returns

This API can return this value:

long int Total number of records currently in the cache.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
```



```
memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
...

/*initialize cache*/
...

/*add some records*/
...

if (jdeCacheGetNumRecords(hCache)<1)
      printf("No records was added to cache[%s]\n",szCacheName);
/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheInit

Syntax

```
JDECM_RESULT jdeCacheInit ⇒
(HUSER hUser, HCACHE *hCache, char * szCacheName, JDECMINDEXSTRUCT *lpIndex);
```

Description

jdeCacheInit initializes a named cache, associates the initialized cache with a user and returns a handle to the initialized cache. This API:

1. Creates a cache in memory.
2. Creates an index that will be used to access records in the cache.
3. Names the cache with the name passed with *szCacheName*.
4. Associates the cache with a cache handle (HCACHE) that the user will use to reference the cache in call to JDE Cache APIs.

You can initialize more than one cache handle using the same cache name, but the *lpIndex* structure must be an exact match. **jdeCacheInit** creates a cache with 100 cursors. To create a cache and define the number of cursors, use **jdeCacheInitEx**.

Note. Without a call to this API or to **jdeCacheInitEx**, none of the JDECache APIs will work.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The created cache is associated with this handle.
hCache	Output. This cache handle identifies the cache on every call to a JDE Cache API. This means that if a NULL is returned by this API, then any subsequent call to the operational JDE Cache APIs will fail. It is always the first parameter to all the other JDE Cache APIs except for this API and **jdeCacheTerminate**. There is no limit for the number of cache handles that can be initialized. Do not reuse this handle without having called a **jdeCacheTerminate**; otherwise, a memory leak will occur.
szCacheName	Input, required. A unique cache identifier. If there is more than one cache, they all should have distinct names. *szCacheName* cannot be longer than 45 characters.
lpIndex	Input. This index references the data in the cache. It must point to one and only one JDECMINDEXSTRUCT. There cannot be more than 25 *CacheKey*s in one *lpIndex*.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
if (jdeCacheInit(hUser, &hCache, szCacheName, Index)==JDECM_FAILED)
{
      printf("Initialize cache failed \n");
      /*do something*/
      return ;
```

```
}

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheInitEx

Syntax

```
JDECM_RESULT jdeCacheInitEx ⇒
(HUSER hUser, HCACHE * hCache, JCHAR * szCacheName, JDECMINDEXSTRUCT *lpIndex,⇒
short int nSizeCursors);
```

Description

jdeCacheInitEx initializes a named cache, associates the initialized cache with a user and returns a handle to the initialized cache. This API:

1. Creates a cache in memory.
2. Creates an index that will be used to access records in the cache.
3. Names the cache with the name passed with *szCacheName*.
4. Associates the cache with a cache handle (HCACHE) that the user will use to reference the cache in call to JDE Cache APIs.
5. Enables the caller to pass the maximum number of cursors within the range 1 to 100.

You can initialize more than one cache handle using the same cache name, but the *lpIndex* structure must be an exact match.

Note. Without a call to this API or to **jdeCacheInit**, none of the JDECache APIs will work.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The created cache is associated with this handle.
hCache	Output. This cache handle identifies the cache on every call to a JDE Cache API. This means that if a NULL is returned by this API, then any subsequent call to the operational JDE Cache APIs will fail. It is always the first parameter to all the other JDE Cache APIs except for this API and **jdeCacheTerminate**. There is no limit for the number of cache handles that can be initialized. Do not reuse this handle without having called a **jdeCacheTerminate**; otherwise, a memory leak will occur.
szCacheName	Input, required. A unique cache identifier. If there is more than one cache, they all should have distinct names. *szCacheName* cannot be longer than 45 characters.
lpIndex	Input. This index references the data in the cache. It must point to one and only one JDECMINDEXSTRUCT. There cannot be more than 25 *CacheKey*s in one *lpIndex*.
nSizeCursors	Input. The maximum number of cursors per cache. It must be in the range from 1 to 100.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
short int nMaxCursors = 10;

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 30;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;
```



```
/*initialize cache*/
if (jdeCacheInit(hUser, &hCache, szCacheName, Index, nMaxCursors ) == JDECM_FAILED⇒
 )
{
printf("Initialize cache failed \n");
 /*do something*/
 return ;
}

/*do something*/
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheInitMultipleIndex

Syntax

```
JDECM_RESULT jdeCacheInitMultipleIndex⇒
(HUSER hUser, HCACHE * hCache, char * szCacheName,⇒
JDECMINDEXSTRUCT *lpIndex, short int nNumindexes)
```

Description

jdeCacheInitMultipleIndex initializes a named multiple indexes cache, associates the initialized cache with a user and returns a handle to the initialized cache. This API accomplishes four things:

1. Creates a cache in memory.
2. Creates indexes that will be used to access records in the cache.
3. Names the cache with the name passed with *szCacheName*.
4. Associates the cache with a cache handle (HCACHE) that the user will use to reference the cache in call to JDE Cache APIs.

More than one cache handle can be initialized by using the same cache name, but the lpIndex structure must be an exact match.

Note. Without a call to this API or to **jdeCacheInitMultipleindexes**, none of the JDECache API will work.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The created cache is associated with this handle.
hCache	Output. This cache handle identifies the cache on every call to a JDE Cache API. This means that if a NULL is returned by this API, then any subsequent call to the operational JDE Cache APIs will fail. It is always the first parameter to all the other JDE Cache APIs except for this API and **jdeCacheTerminate**. There is no limit for the number of cache handles that can be initialized. Do not reuse this handle without having called a **jdeCacheTerminate**; otherwise, a memory leak will occur.
szCacheName	Input, required. A unique cache identifier. If there is more than one cache, they all should have distinct names. *szCacheName* cannot be longer than 45 characters.
lpIndex	Input. This index references the data in the cache. It must point to one and only one JDECMINDEXSTRUCT. There cannot be more than 25 *CacheKey*s in one *lpIndex*.
nNumindexes	Input. Total number of indexes declared in *lpIndex*.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[2];
short int nNumindexes   =  2;
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up multiple indexes structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT)*2);
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

Index[1].nKeyID = 2;
Index[1].nNumSegments = 1;
Index[1].CacheKey[0].nOffset = 10;
```

```
Index[1].CacheKey[0].nSize=5;
Index[1].CacheKey[0].idDataType = EVDT_STRING;

/*initialize multiple indexes*/
if (jdeCacheInitMultipleIndex(hUser, &hCache, szCacheName, Index, nNumindexes)==⇒
JDECM_FAILED)
{
      printf("Initialize cache failed \n");
      /*do something*/
      return ;
}

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheInitMultipleindexEx

Syntax

```
JDECM_RESULT jdeCacheInitMultipleindexes⇒
(HUSER hUser, HCACHE * hCache, char * szCacheName, JDECMINDEXSTRUCT *lpIndex, ⇒
short int nNumindexes, short int nSizeCursors);
```

Description

jdeCacheInitMultipleindexes initializes a named multiple indexes cache, associates the initialized cache with a user and returns a handle to the initialized cache. This API accomplishes four things:

1. Creates a cache in memory.
2. Creates indexes that will be used to access records in the cache.
3. Names the cache with the name passed with *szCacheName*.
4. Associates the cache with a cache handle (HCACHE) that the user will use to reference the cache in call to JDE Cache APIs.
5. Enables the caller to pass the maximum number of cursors within the range 1 to 100.

More than one cache handle can be initialized by using the same cache name, but the lpIndex structure must be an exact match.

Note. Without a call to this API or to **jdeCacheInitMultipleIndex**, none of the JDECache API will work.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The created cache is associated with this handle.
hCache	Output. This cache handle identifies the cache on every call to a JDE Cache API. This means that if a NULL is returned by this API, then any subsequent call to the operational JDE Cache APIs will fail. It is always the first parameter to all the other JDE Cache APIs except for this API and **jdeCacheTerminate**. There is no limit for the number of cache handles that can be initialized. Do not reuse this handle without having called a **jdeCacheTerminate**; otherwise, a memory leak will occur.
szCacheName	Input, required. A unique cache identifier. If there is more than one cache, they all should have distinct names. *szCacheName* cannot be longer than 45 characters.
lpIndex	Input. This index references the data in the cache. It must point to one and only one JDECMINDEXSTRUCT. There cannot be more than 25 *CacheKey*s in one *lpIndex*.
nNumindexes	Input. Total number of indexes declared in *lpIndex*.
nSizeCursors	Input. The maximum number of cursors per cache. It must be in the range from 1 to 100.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[2];
char szCacheName[20];
short int nMaxCursors = 10;
short totalIndexes = 2

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT)*2);
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 30;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;
```



```
Index[1].nKeyID = 2;
Index[1].nNumSegments = 2;
Index[1].CacheKey[0].nOffset = 50;
Index[1].CacheKey[0].nSize=10;
Index[1].CacheKey[0].idDataType = EVDT_STRING;
Index[1].CacheKey[1].nOffset = 80;
Index[1].CacheKey[1].nSize=10;
Index[1].CacheKey[1].idDataType = EVDT_STRING;

  /*initialize cache*/
if (jdeCacheInitMultipleindexes(hUser, &hCache, szCacheName, Index , totalIndexes,⇒
 nMaxCursors ) == JDECM_FAILED )
{
printf("Initialize cache failed \n");
 /*do something*/
 return ;
}

/*do something*/
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheInitMultipleIndexUser

Syntax

```
JDECM_RESULT jdeCacheInitMultipleIndexUser⇒
(HUSER hUser, HCACHE * hCache, char * szCacheName,JDECMINDEXSTRUCT *lpIndex, ⇒
short int nNumindexes, LPBHVRCOM lpBhvrCom)
```

Description

jdeCacheInitMultipleIndexUser initializes a named multiple indexes cache, associates the initialized cache with a user and returns a handle to the initialized cache. This API accomplishes five things:

1. Creates a cache in memory.
2. Creates indexes that will be used to access records in the cache.
3. Makes the cache name to be unique per application.
4. Names the cache with the created unique name.
5. Associates the cache with a cache handle (HCACHE) that the user will use to reference the cache in call to JDE Cache APIs.

More than one cache handles can be initialized by using the same cache name, but the *lpIndex* structure must be an exact match.

Note. Without a call to this API, none of the jdeCache APIs will work

---

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The created cache is associated with this handle.
hCache	Output. This cache handle identifies the cache on every call to a JDE Cache API. This means that if a NULL is returned by this API, then any subsequent call to the operational JDE Cache APIs will fail. It is always the first parameter to all the other JDE Cache APIs except for this API and **jdeCacheTerminate**. There is no limit for the number of cache handles that can be initialized. Do not reuse this handle without having called a **jdeCacheTerminate**; otherwise, a memory leak will occur.
szCacheName	Input, required. A unique cache identifier. If there is more than one cache, they all should have distinct names. *szCacheName* cannot be longer than 45 characters.
lpIndex	Input. This index references the data in the cache. It must point to one and only one JDECMINDEXSTRUCT. There cannot be more than 25 *CacheKey*s in one *lpIndex*.
nNumindexes	Input. Total number of indexes declared in *lpIndex*.
lpBhvrCom	Input. Pointer to behavior com structure. This structure will provide cache with the unique manual commit *hUuser* address used to make cache unique per application.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[2];
short int nNumindexes   =  2;
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up multiple indexes structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT)*2);
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

Index[1].nKeyID = 2;
```

```
Index[1].nNumSegments = 1;
Index[1].CacheKey[0].nOffset = 10;
Index[1].CacheKey[0].nSize=5;
Index[1].CacheKey[0].idDataType = EVDT_STRING;

/*initialize multiple indexes*/
if (jdeCacheInitMultipleIndexUser(hUser, &hCache, szCacheName, Index, nNumindexes,⇒
 lpBhvrCom)==JDECM_FAILED)
{
       printf("Initialize cache failed \n");
      /*do something*/
      return ;
}

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheInitUser

Syntax

```
JDECM_RESULT jdeCacheInitUser⇒
(HUSER hUser, HCACHE * hCache, char * szCacheName , JDECMINDEXSTRUCT * lpIndex, ⇒
LPBHVRCOM lpBhvrCom)
```

Description

jdeCacheInitUser initializes a named cache, associates the initialized cache with a user and returns a handle to the initialized cache. This API accomplishes five things:

1. Creates a cache in memory
2. Creates an index that will be used to access records in the cache.
3. Makes the cache name to be unique per application.
4. Names the cache with the created unique name.
5. Associates the cache with a cache handle (HCACHE) that the user will use to reference the cache in call to JDE Cache APIs.

More than one cache handles can be initialized by using the same cache name, but the *lpIndex* structure must be an exact match.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The created cache is associated with this handle.
hCache	Output. This cache handle identifies the cache on every call to a JDE Cache API. This means that if a NULL is returned by this API, then any subsequent call to the operational JDE Cache APIs will fail. It is always the first parameter to all the other JDE Cache APIs except for this API and **jdeCacheTerminate**. There is no limit for the number of cache handles that can be initialized. Do not reuse this handle without having called a **jdeCacheTerminate**; otherwise, a memory leak will occur.
szCacheName	Input, required. A unique cache identifier. If there is more than one cache, they all should have distinct names. *szCacheName* cannot be longer than 45 characters.
lpIndex	Input. This index references the data in the cache. It must point to one and only one JDECMINDEXSTRUCT. There cannot be more than 25 *CacheKey*s in one *lpIndex*.
lpBhvrCom	Input. Pointer to behavior com structure. This structure will provide cache with the unique manual commit *hUuser* address used to make cache unique per application.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/

memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
if (jdeCacheInitUser(hUser, &hCache, szCacheName, Index, lpBhvrCom)==JDECM_FAILED)
```



```
{
printf("Initialize cache failed \n");
      /*do something*/
      return ;
}

/*do something*/

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)

     printf("jdeCacheTerminate Failed \n");
```

jdeCacheOpenCursor

Syntax

```
JDECM_RESULT jdeCacheOpenCursor (HCACHE hCache, HJDECURSOR * hCursor);
```

Description

jdeCacheOpenCursor opens a cursor for use. Without opening a cursor, all calls to the JDE Cache Data Manipulating APIs will fail. When a cursor is opened, the cache is first checked to see if it contains any records. If it does then the number of cursors for this cache is checked. If the number is less than the maximum permitted, a new cursor is created and added to the JDE Cache Cursor Manager. This cursor is then assigned to be the current ACTIVE HJDECURSOR for the HCACHE.

Note. The maximum number of open cursors on any one cache is 100

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hCache*. Use this cursor handle used in calls to Data Manipulation APIs to reference records in the cache.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
```

```
memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
...

/*initialize cache*/
...
/*add some records*/
...

/*open a cursor*/
if (jdeCacheOpenCursor(hCache,&hCursor)==JDECM_FAILED)
     printf("Open cursor failed !\n")
...

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheResetCursor

Syntax

```
JDECM_RESULT jdeCachResetCursor (HCACHE hCache, HJDECURSOR hCursor);
```

Description

jdeCacheResetCursor resets a cursor, for example, it repositions the cursor to point to the first record in the data set using the first index. The passed cursor is assigned to be the ACTIVE HJDECURSOR cursor for the cache, and if the cache is not empty, then the repositioning takes place.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hCache*. Use this cursor handle used in calls to Data Manipulation APIs to reference records in the cache.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
...

/*initialize cache*/
...

/*add some records*/
...

/*open a cursor*/
...

/*do something*/
...

if (jdeCacheGetNumRecords(hCache)>0)
{
      if (jdeCacheResetCursor(hCache,hCursor)==JDECM_FAILED)
                printf("jdeCacheResetCursor Failed, something very bad happened⇒
\n");
}
else
    printf("The cache is empty and there is no record for the cursor to point at⇒
\n");

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheSetIndex

Syntax

```
JDECM_RESULT jdeCacheSetIndex(HCACHE hCache, HJDECURSOR hCursor, long int nIndexID)
```


Description

jdeCacheSetIndex enables you to set a cursor point to a desired index field by passing in the index ID. Use this API when you are using multiple indexes.

Parameters

Parameter	Description
hCache	Input. A cache handle that uniquely identifies the cache.
hCursor	Input. The cursor pointing to the data set within the cache associated with *hCache*. Use this cursor handle used in calls to Data Manipulation APIs to reference records in the cache.
nIndexID	Input. One of the declared Index IDs in the Index structure.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[2];
short int nNumindexes   =  2;
char szCacheName[20];
long int nKeyID = 2;

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up multiple indexes structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT)*2);
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

Index[1].nKeyID = 2;
Index[1].nNumSegments = 1;
Index[1].CacheKey[0].nOffset = 10;
Index[1].CacheKey[0].nSize=5;
Index[1].CacheKey[0].idDataType = EVDT_STRING;
```



```
/*initialize multiple indexes*/
if (jdeCacheInitMultipleIndex(hUser, &hCache, szCacheName, Index, nNumindexes)==⇒
JDECM_FAILED)
{
      printf("Initialize cache failed \n");

      /*do something*/
      return;
}

/*add some records*/
...

/*open cursor*/
...

if (jdeCacheSetIndexNumCursors(hCache,hCursor,nKeyID)==JDECM_FAILED)
      printf("Set Index ID failed !!\n");

/*do something*/
...

if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheTerminate

Syntax

```
JDECM_RESULT jdeCacheTerminate (HUSER hUser,HCACHE hCache)
```

Description

jdeCacheTerminate dissociates a named cache handle from the cache. This means that the passed cache handle cannot be used to access this cache anymore. This has the effect of destroying the cache being referenced by the cache handle to the calling application. If this is the **jdeCacheTerminate** call that matches the total number of **jdeCacheInit** calls, then the cache is not only disassociated from the cache handle but also physically destroyed from memory.

Note. If more than one cache handle with the same cache name is initialized, one call to this API does not terminate all of the cache handles from memory. You can terminate all of them by calling **jdeCacheTerminateAll**.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The destroyed cache is that associated with this handle.
hCache	Output. A cache handle that uniquely identifies the cache. After **jdeCacheTerminate** returns, this cache cannot be accessed. You must start the caching process again to access this cache.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/

memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
if (jdeCacheInit(hUser, &hCache, szCacheName, Index)==JDECM_FAILED)
{
      printf("Initialize cache failed \n");

      /*do something*/
      return ;
}

/*do something*/
...
```



```
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

jdeCacheTerminateAll

Syntax

```
JDECM_RESULT jdeCacheTerminateAll (HUSER hUser, HCACHE hCache)
```

Description

jdeCacheTerminateAll destroys all caches with the same name as the one with the given *hCache*. The caches are physically removed from memory.

Parameters

Parameter	Description
hUser	Input. The user handle obtained from a call to the JDB_ API. It cannot be NULL. The destroyed caches are those associated with this handle.
hCache	Output. A cache handle that uniquely identifies the cache. After **jdeCacheTerminateAll** returns, these caches cannot be accessed. You must start the caching process again to access this cache.

Returns

This API can return these values:

JDECM_PASSED	Returned when this API succeeds.
JDECM_FAILED	Returned when this API fails.

Example

```
HCACHE hCache;
HCACHE hCache1;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char szCacheName1[20];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");
memset(szCacheName1,0x00,sizeof(szCacheName1));
strcpy(szCacheName,"B4321234CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
ndex[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
```

```
Index[0].CacheKey[1].nOffset = 15;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
if (jdeCacheInit(hUser, &hCache, szCacheName, Index)==JDECM_FAILED)
{
printf("Initialize cache failed \n");
      /*do something*/
      return ;
}
...
if (jdeCacheInit(hUser, &hCache1, szCacheName1, Index)==JDECM_FAILED)
{
printf("Initialize cache failed \n");
      /*do something*/
      return ;
}
...
if (jdeCacheInit(hUser, &hCache, szCacheName, Index)==JDECM_FAILED)
{
printf("Initialize cache failed \n");
      /*do something*/
      return ;
}
/*do something*/
...
/*terminates all caches*/
if (jdeCacheTerminateAll(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminateAll Failed \n");
```

jdeCacheUpdate

Syntax

```
JDECM_RESULT jdeCacheUpdate⇒
(HCACHE hCache, HJDECURSOR hCursor, void *lpData, long nDataSize);
```

Description

jdeCacheUpdate updates the record that is being pointed to by the passed HJDECURSOR with the passed data. You should use a cursor manipulation API that does not advance the cursor, such as **jdeCacheFetch** or **jdeCacheFetchPosition** to position the cursor to the record to be updated before calling this API.

If the cache is not empty and the cursor is open, then the record currently being pointed to by HJDECURSOR is updated.

Parameters

Parameter	Description
hCache	Intput. A cache handle that uniquely identifies the cache.
hCursor	Input. Use this cursor handle used in calls to Data Manipulation APIs to reference records in the cache.
lpData	Input. *ndPointer* structure containing the data to be used to update the cache.
nDataSize	Input. The size of the structure that is being passed with the data.

Returns

This API can return these values:

JDECM_PASSED Returned when this API succeeds.

JDECM_FAILED Returned when this API fails.

Example

```
HCACHE hCache;
HJDECURSOR hCursor;
JDECMINDEXSTRUCT Index[1];
char szCacheName[20];
char Data[100];
char ret_Data[100];

memset(szCacheName,0x00,sizeof(szCacheName));
strcpy(szCacheName,"B3200052CMCache");

/*set up index structure*/
memset(Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index[0].nKeyID = 1;
Index[0].nNumSegments = 2;
Index[0].CacheKey[0].nOffset = 0;
Index[0].CacheKey[0].nSize=10;
Index[0].CacheKey[0].idDataType = EVDT_STRING;
Index[0].CacheKey[1].nOffset = 10;
Index[0].CacheKey[1].nSize=10;
Index[0].CacheKey[1].idDataType = EVDT_STRING;

/*initialize cache*/
...
/*add  records*/
strcpy(Data,"Testing");
strcpy(Data+10, "12345");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");

memset(Data,0x00,sizeof(Data));
```



```
strcpy(Data,"Testing");
strcpy(Data+10, "67890");
if (jdeCacheAdd(hCache,Data,sizeof(Data))==JDECM_FAILED)
      printf("Add record failed !!\n");

/*open a cursor*/
...

/*fetch position the cursor to the first record that has the key "Testing  ⇒
 67890"*/
memset(ret_Data,0x00,sizeof(ret_Data));
if (jdeCacheFetchPosition(hCache,hCursor,Data,2,ret_Data,sizeof(ret_Data))==JDECM_⇒
FAILED)
     printf("Record not found\n");

/*update the located record*/
memset(Data,0x00,sizeof(Data));
strcpy(Data,"DoneTest");
strcpy(Data+10, "45612");
if (jdeCacheUpdate(hCache,hCursor,Data,sizeof(Data))==JDECM_FAILED)
     printf("jdeCacheUpdate Failed\n");
/*do something*/
...
if (jdeCacheTerminate(hUser,hCache)==JDECM_FAILED)
     printf("jdeCacheTerminate Failed \n");
```

JD Edwards EnterpriseOne Threadsafe APIs

This section discusses the JD Edwards EnterpriseOne threadsafe APIs that you use to replace non-threadsafe standard C APIs.

jdePPSRand

Syntax

```
void jdePPSRand(int seed)
```

Description

This is a replacement for the standard C API **srand()**, to be used in conjunction with the API **jdePPRand**. The **jdePPSRand** function sets the seed at the process level for a new sequence of pseudo-random numbers to be returned by subsequent calls to **jdePPRand**. If **jdePPSRand** is called with the same seed value, the same sequence of pseudo-random numbers is repeated by **jdePPRand**. If the function **jdePPSRand** is not called to initialize the seed, the first call to **jdePPRand** returns a random number with seed equal to 1 and then creates a new seed stored.

Parameters

This API takes the following parameters, which are the same as those for the standard C API **srand**:

Parameter	Description
Seed	Input, required. Used to guarantee unique, pseudo-random numbers generated by jdePPRand. Must be an integer greater than zero.

Returns

This API has no return values.

Example

This example creates a random number with a seed which is unique for each process. You should call **jdePPSRand** only once to initialize a seed. After each call to the threadsafe function **jdePPRand**, a new seed is created and stored internally in the global variable.

```
void foo(int * rand)
 {
  jdePPSRand((unsigned)time( NULL ));
  *rand=jdePPRand();
 }
```

jdePPRand

Syntax

```
int jdePPRand(void);
```

Description

This is a threadsafe replacement for the standard C API **rand()**. **jdePPRand** returns an unique pseudo-random integer between 0 and RAND_MAX for multithreading process. If the function **jdePPSRand** is not called to initialize the seed, the first call to the function **jdePPRand** returns a random number with seed equal to 1 and then creates a new seed stored. Any following call to the function **jdePPRand** would use the seed created previously to return a random number and then update the seed.

Parameters

This API takes no parameters.

Returns

This API can return this value:

An integer	An pseudo-random integer value greater than zero.

Example

This example creates a random number without setting the first seed.

```
void foo(int *rand)
  {
   *rand=jdePPRand();
  }
```

jdeLocaltime

Syntax

```
void jdeLocaltime(const time_t *pnTime, struct tm *pzTime)
```

Description

This is a threadsafe replacement for the standard C API **localtime()**. It converts the time passed in by *pnTime*, to a local time value. This local time is returned in the *pzTime* buffer parameter allocated and passed in by the caller.

Parameters

This API can take these parameters:

Parameter	Description
PnTime	A pointer to a *time_t* structure containing value to convert to local time.
PzTime	Return parameter to store converted local time value.

Returns

This API has no return values.

Example

```
Void foo()
{ struct tm  today     = {0};
 struct _timeb ztimeBuffer   = {0};
 _ftime( &ztimeBuffer );
 jdeLocaltime(&ztimeBuffer.time, &today);
}
```

Media Object APIs

This section discusses media object APIs.

jdeGT_CloseTable

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_CloseTable(HREQUEST hRequestGT);
```

Description

This function closes the F00165 table and releases the table handle. This API must be invoked after the **jdeGT_OpenTable()** API is used.

Parameters

Parameter	Description
hRequestGT	Input, required. GT table handle to be closed and released.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("");

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
...
...
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 163
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 167
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 172
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_OpenTable, page 176
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 178
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182

jdeGT_DeleteData/jdeGT_DeleteDataKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_DeleteData(
        HREQUEST hRequestGT,
```



```
          PJSTR pszOjbectName,
          LPVOID lpMODSKey,
          int nSeq
          );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_DeleteDataKeyStr(
          HREQUEST hRequestGT,
          PJSTR pszOjbectName,
          PJSTR pszGTKeyStr,
          int nSeq
          );
```

Description

This function deletes a record in the F00165 table.

Parameters

Parameter	Description
hRequestGT	Input, required. GT table handle to be closed and released.
pszObjectName	Input, required. GT data structure name.
lpMODSKey	Input, required. GT data structure with data loaded (use in **jdeGT_SelectData**). This data is formatted into the string for TXKY.
pszGTKeyStr	Input, required. GT-formatted string from the GT data structure (use in **jdeGT_SelectDataKeyStr**).

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Additional Notes

Invoke this API after using the **jdeGT_OpenTable()** API. This API is used for multiple access of the table within one function scope.

This table describes the MODATA (or LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.

Data Type	Data Description	Note
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Example

This is the first of two examples for this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
JCHAR szFormatKey[255] = _J("1");
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
```



```
 JDBReturn = jdeGT_SelectDataKeyStr (hRequest, szObjectName, szFormatKey, sz⇒
Lang,
                 OBJ_JDEOLE);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 while (JDBReturn = JDEDB_PASSED)
 {
  jdeGT_DeleteDataKeyStr(hRequest, szObjectName, szFormatKey, lpGTData->nSeq);
  jdeGTFreeMOData(lpGTData, 1);
  JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 }
}

jdeGTFreeMOData(lpGTData, 1);
jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

This is the second example for this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
DSABGT dsAbGT = {0};
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;

JDBReturn = JDB_InitBhvr(.., &hUser);

ParseNumericString(dsAbGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectData(hRequest, szObjectName, &dsAbGT, szLang, OBJ_⇒
JDEOLE);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 while (JDBReturn = JDEDB_PASSED)
 {
  jdeGT_DeleteData(hRequest, szObjectName, &dsAbGT, lpGTData->nSeq);
```



```
      jdeGTFreeMOData(lpGTData, 1);
      JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
     }
    }

    jdeGTFreeMOData(lpGTData, 1);
    jdeGT_CloseTable(hRequestGT);
    JDB_FreeBhvr(hUser);

    return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_CloseTable, page 162
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 167
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 172
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_OpenTable, page 176
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 178
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182

jdeGT_FetchData/jdeGT_FetchDataEx

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_FetchData(
         HREQUEST hRequestGT,
         LPMODATA *lpMOData
         BOOL bConvRTFText
         );

JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_FetchDataEx(
         HREQUEST hRequestGT,
         JCHAR *szObjectName,
         JCHAR *szGTKey,
         LPMODATA *lpMOData
         BOOL bConvRTFText
         );
```

Description

This function retrieves one record from the F00165 table.

Parameters

Parameter	Description
hRequestGT	Input, required. GT table handle to be closed and released.
szObjectName	Output. Object name (OBNM)-character array.
szGTKey	Output. Generic text key in the string format (TXKY)-character array.
**lpMOData*	Output. Allocated memory for data.
bCovRTFText	Input. TRUE = Convert any RTF text to plain text.

Additional Notes

Invoke this API after using the **jdeGT_OpenTable()** API. This API is used for multiple access of the table with one function scope.

This table describes the MODATA (or LOMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of three examples for this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
JCHAR szFormatKey[255] = _J("1");
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectDataKeyStr (hRequest, szObjectName, szFormatKey, sz⇒
Lang,
                 OBJ_JDEOLE);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 while (JDBReturn = JDEDB_PASSED)
 {
  jdeGT_DeleteDataKeyStr(hRequest, szObjectName, szFormatKey, lpGTData->nSeq);
  jdeGTFreeMOData(lpGTData, 1);
  JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 }
```

```
}

jdeGTFreeMOData(lpGTData, 1);
jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

This is the second example of this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
DSABGT dsAbGT = {0};
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;

JDBReturn = JDB_InitBhvr(.., &hUser);

ParseNumericString(dsAbGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectData(hRequest, szObjectName, &dsAbGT, szLang, OBJ_⇒
JDEOLE);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 while (JDBReturn = JDEDB_PASSED)
 {
  jdeGT_DeleteData(hRequest, szObjectName, &dsAbGT, lpGTData->nSeq);
  jdeGTFreeMOData(lpGTData, 1);
  JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 }
}

jdeGTFreeMOData(lpGTData, 1);
jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

This is the third example of this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
```



```
HUSER hUser = NULL;
LPMODATA lpGTData = NULL;
JCHAR szGTKey[256] = _J("");
JCHAR szObjName[11] = _J("");
JDBReturn = JDB_InitBhvr(lpBhvrCom, &hUser,(JCHAR *) NULL,
           JDEDB_COMMIT_AUTO
);
if (lpDS->idHRequestGT == NULL)
{
 JDBReturn = jdeGT_OpenTable (hUser, lpDS->szFromDatasource,
 lpDS->szObjectName, &hRequestGT);
 lpDS->idHRequestGT = (ID) jdeStoreDataPtr(hUser, hRequestGT);
else
{
 hRequestGT = (HREQUEST) jdeRetrieveDataPtr(hUser, lpDS->idHRequestGT);
if (hRequstGT)
{
 JDBReturn = jdeGT_SelectData(hRequest, lpDS->szObjectName,
    NULL, lpDS->szLang, OBJ_JDEOLE);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchDataEx(hRequest, szObjName, szGTKey, lpGTData, TRUE);
if (JDBReturn = JDEDB_PASSED)
{
 jdeStrncpy(lpDS->szGTKeyData, szGTKey, 255);
 jdeStrncpy(lpDS->szTextData, lpGTData->pData, 255);
 jdeGTFreeMOData(lpGTData, 1);
 else
 {
 jdeRemoveDataPtr(hUser, lpDS->idHRequestGT);
 lpDS->idHRequestGT = 0L;
 }
}
else
{
 jdeRemoveDataPtr(hUser, lpDS->idHRequestGT);
 lpDS->idHRequestGT = 0L;
}
 if (hRequestGT && lpDS->idHRequestGT == 0)
 {
 jdeGT_CloseTable(hRequestGT);
 }

JDB_FreeBhvr(hUser);
return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_CloseTable, page 162
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 163
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 172
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_OpenTable, page 176
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 178
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182

jdeGT_InsertData/jdeGT_InsertDataKeyStr

Syntax

```
JDERTN(JDEDB_RESULT)JDEWINAPI jdeGT_InsertData(
         HREQUEST hRequestGT,
         PJSTR pszOjbectName,
         LPVOID lpMODSKey,
         int nSeq,
         PJSTR pszGTLang,
         LPMODATA lpMOData
         );

JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_InsertDataKeyStr(
         HREQUEST hRequestGT,
         PJSTR pszOjbectName,
         PJSTR pszGTKeyStr,
         int nSeq,
         PJSTR pszGTLang,
         LPMODATA lpMOData
         );
```

Description

This function inserts a record into the F00165 table.

Parameters

Parameter	Description
hRequestGT	Input, required. GT table handle to be closed and released.
pszObjectName	Input, required. GT data structure name.
lpMODSKey	Input, required. GT data structure with data loaded (**use in jdeGT_SelectData**). This data is formatted into the string for TXKY.
pszGTKeyStr	Input, required. GT formatted string from GT data structure (use in **jdeGT_SelectDataKeyStr**).
nSeq	Input, required. Sequence number for primary key.
pszGTLang	Input. Language code to be updated.
lpMOData	Input, required. Data to be updated.

Additional Notes

Invoke this API after using the **jdeGT_OpenTable()** API. This API is used for multiple access of the table with one function scope.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples for this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
JCHAR szFormatKey[255] = _J("1");
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;
MODATA dsGTNewData = {0};
JCHAR szText[255] = _J("New Text to be inserted");

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectDataKeyStr (hRequest, szObjectName, szFormatKey, sz⇒
Lang,
                 OBJ_JDEALL);
}

if (JDBReturn == JDEDB_PASSED)
```

```
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 if (JDBReturn != JDEDB_PASSED)
 {
  dsGTNewData.nSeq = 1;
  dsGTNewData.nMOType = OBJ_RTFTEXT;
  dsGTNewData.pData = szText;
  jdeStrcpy(dsGTNewData.szItemName, _J("New Text"));
  jdeGT_InsertDataKeyStr(hRequest, szObjectName, szFormatKey, 1, szLang, &ds⇒
GTNewData);
 }
 else
 {
  jdeGTFreeMOData(lpGTData, 1);
 }
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

This is the second example for this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
DSABGT dsAbGT = {0};
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;
MODATA dsGTNewData = {0};
JCHAR szText[255] = _J("New Text to be inserted");


JDBReturn = JDB_InitBhvr(.., &hUser);

ParseNumericString(dsAbGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectData(hRequest, szObjectName, &dsAbGT, szLang, OBJ_⇒
JDEALL);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
```



```
 if (JDBReturn != JDEDB_PASSED)
 {
  dsGTNewData.nSeq = 1;
  dsGTNewData.nMOType = OBJ_RTFTEXT;
  dsGTNewData.pData = szText;
  jdeStrcpy(dsGTNewData.szItemName, _J("New Text"));
  jdeGT_InsertData(hRequest, szObjectName, &dsAbGT, 1, szLang, &dsGTNewData);
 }
 else
 {
  jdeGTFreeMOData(lpGTData, 1);
 }
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_CloseTable, page 162
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 163
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_FetchData/jdeGT_FetchDataEx, page 167
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_OpenTable, page 176
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 178
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182

jdeGT_OpenTable

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_OpenTable(
         HUSER hUser,
         PJSTR pszDataSource,
         PJSTR pszObjectName,
         HREQUEST *hRequestGT
         );
```

Description

This function enables the F00165 table to be opened based on the object name or data source. It must be used first if the related functions are to be used. This API is used for multiple access of the table within one function scope.

Parameters

Parameter	Description
hUser	Input, required. User handle.
pszDataSource	Input. If empty, use the default data source from the *pszObejctName*. Data source has precedence over the object name.
pszObjectName	Input, required if *pszDatasource* is empty.
hRequestGT	Output. If open table fails, NULL pointer is returned; otherwise, a pointer to the handle is returned.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples for this API, and it demonstrates using data source:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("");

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
...
...
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

This example demonstrates using object name:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J(" ");
```

```
JCHAR szObjectName[11] = _J("ABGT");

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
...
...
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_OpenTable, page 176
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 163
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_FetchData/jdeGT_FetchDataEx, page 167
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 172
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 178
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182

jdeGT_SelectData/jdeGT_SelectDataKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_SelectData(
         HREQUEST hRequestGT,
         PJSTR pszOjbectName,
         LPVOID pdsGTKeyDS,
         PJSTR pszLang,
         MOTYPE nMOType
         );

JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_SelectDataKeyStr(
         HREQUEST hRequestGT,
         PJSTR pszOjbectName,
         PJSTR pszGTKeyStr,
         PJSTR pszLang,
         MOTYPE nMOType
         );
```

Description

This function enables a data selection to be applied against the F00165 table.

Parameters

Parameter	Description
hRequestGT	Input, required. GT table handle to be closed and released.
pszObjectName	Input, required. GT data structure name.
pszLang	Input. Language code.
nMOType	Input, required. Media object type.

Additional Notes

Invoke this API after using the **jdeGT_OpenTable()** API. This API is used for multiple access of the table with one function scope.

This table describes the MODATA (or LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.

Define Type	Note
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples describing how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
JCHAR szFormatKey[255] = _J("1");
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;

JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectDataKeyStr (hRequest, szObjectName, szFormatKey, sz⇒
Lang,
                 OBJ_JDEOLE);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 while (JDBReturn = JDEDB_PASSED)
 {
  jdeGT_DeleteDataKeyStr(hRequest, szObjectName, szFormatKey, lpGTData->nSeq);
  jdeGTFreeMOData(lpGTData, 1);
  JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 }
}
```



```
jdeGTFreeMOData(lpGTData, 1);
jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
DSABGT dsAbGT = {0};
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;

JDBReturn = JDB_InitBhvr(.., &hUser);

ParseNumericString(dsAbGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectData(hRequest, szObjectName, &dsAbGT, szLang, OBJ_⇒
JDEOLE);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 while (JDBReturn = JDEDB_PASSED)
 {
  jdeGT_DeleteData(hRequest, szObjectName, &dsAbGT, lpGTData->nSeq);
  jdeGTFreeMOData(lpGTData, 1);
  JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 }
}

jdeGTFreeMOData(lpGTData, 1);
jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_CloseTable, page 162
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 163
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_FetchData/jdeGT_FetchDataEx, page 167
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 172
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_OpenTable, page 176
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182

jdeGT_UpdateData/jdeGT_UpdateDataKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_UpdateData(
        HREQUEST hRequestGT,
        PJSTR pszOjbectName,
        LPVOID lpMODSKey,
        int nSeq,
        PJSTR pszGTLang,
        LPMODATA lpMOData
        );

JDERTN(JDEDB_RESULT) JDEWINAPI jdeGT_UpdateDataKeyStr(
        HREQUEST hRequestGT,
        PJSTR pszOjbectName,
        PJSTR pszGTKeyStr,
        int nSeq,
        PJSTR pszGTLang,
        LPMODATA lpMOData
        );
```

Description

This function updates an existing record in the F00165 table.

Parameters

Parameter	Description
hRequestGT	Input, required. GT table handle to be closed and release.
pszObjectName	Input, required. GT data structure name.
lpMODSKey	Input, required. GT data structure with data loaded (use in **jdeGT_SelectData**). This data is formatted into the string for TXKY.
nSeq	Input, required. Sequence number for primary key.
pszGTLang.	Input. Language code to be updated.
lpMOData	Input, required. Data to be updated.

Additional Notes

Invoke this API after using the **jdeGT_OpenTable()** API. This API is used for multiple access of the table with one function scope.

This table describes the MODATA (or LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples describing how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
JCHAR szFormatKey[255] = _J("1");
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;


JDBReturn = JDB_InitBhvr(.., &hUser);

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectDataKeyStr (hRequest, szObjectName, szFormatKey, sz⇒
Lang,
                 OBJ_RTFTEXT);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 if (JDBReturn == JDEDB_PASSED)
 {
  jdeFree(lpGTData->pData);

  lpGTData->pData = jdeAlloc(COMMON_POOL, 255*sizeof(JCHAR), MEM_ZEROINIT);
  jdeStrcpy(lpGTData->pData, _J("New Text to be inserted");
  jdeGT_UpdateDataKeyStr(hRequest, szObjectName, szFormatKey, lpGTData->nSeq,⇒
 szLang,
            lpGTData);
  jdeGTFreeMOData(lpGTData, 1);
 }
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```


This is the second example describing how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
HREQUEST hRequestGT = NULL;
HUSER hUser = NULL;
JCHAR szFromDatasource[51] = _J("Business Data - Adev733o");
JCHAR szObjectName[11] = _J("ABGT");
DSABGT dsAbGT = {0};
JCHAR szLang[3] = _J("");
LPMODATA lpGTData = NULL;


JDBReturn = JDB_InitBhvr(.., &hUser);

ParseNumericString(dsAbGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGT_OpenTable (hUser, szFromDatasource, szObjectName, &hRequestGT);
if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_SelectData(hRequest, szObjectName, &dsAbGT, szLang, OBJ_⇒
RTFTEXT);
}

if (JDBReturn == JDEDB_PASSED)
{
 JDBReturn = jdeGT_FetchData(hRequest, lpGTData, FALSE;
 if (JDBReturn == JDEDB_PASSED)
 {
  jdeFree(lpGTData->pData);

  lpGTData->pData = jdeAlloc(COMMON_POOL, 255*sizeof(JCHAR), MEM_ZEROINIT);
  jdeStrcpy(lpGTData->pData, _J("New Text to be inserted");
  jdeGT_UpdateData(hRequest, szObjectName, &dsAbGT, lpGTData->nSeq, szLang,
         lpGTData);
  jdeGTFreeMOData(lpGTData, 1);
 }
}

jdeGT_CloseTable(hRequestGT);
JDB_FreeBhvr(hUser);

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 182
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 163
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 167
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 172
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_OpenTable, page 176
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 178

jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_AllMOType(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );

JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_AllMOTypeKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec
         );

JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_AllMOTypeWithLang(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         PJSTR szLanguage,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function adds or updates all record types to the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
szLanguage	Input. Language code.
pMOData	Input, required. Array of data structure that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples demonstrating how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_AllMOTypeKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example demonstrating how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_AllMOType(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_HTML(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_HTMLKeyStr(
         PJSTR szObjectName,
```



```
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function adds or updates HTML, URL, and File record types to the F00165 table.

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structures that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_HTMLKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
```



```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_HTML (_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_Image(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_ImageKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec);
```

Description

This function adds or updates image record types to the F00165 table.

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structures that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others

Data Type	Data Description	Note
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_ImageKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}
```



```
return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_Image(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_OLE(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_OLEKeyStr(
         PJSTR szObjectName,
```



```
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function adds or updates OLE record types to the F00165 table.

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structures that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_OLEKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
```



```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_OLE(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_Shortcut (
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_ShortcutKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function adds or updates Shortcut record types to the F00165 table.

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structures that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.

Data Type	Data Description	Note
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
```



```
  jdeGTAddUpdate_ShortcutKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_Shortcut (_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_Text(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        LPMODATA pMOData,
        long lTotalRec
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_TextKeyStr(
        PJSTR szObjectName,
```

```
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function adds or updates text record types to the F00165 table.

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structures that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

The API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_TextKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
```

```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_Text(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
 jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_Vendor(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTAddUpdate_VendorKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function adds or updates vendor or third-party software record types to the F00165 table.

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structures that stores the data to be updated.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

Parameter *pData* in MODATA must contain a valid pointer to update the text or shortcut media object type.

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.

Data Type	Data Description	Note
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

The API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
```

```
  jdeGTAddUpdate_VendorKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTAddUpdate_Vendor (_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllHTML(
         PJSTR szObjectName,
         LPVOID lpMODSKey
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllHTMLKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr
         );
```


Description

This function deletes all HTML, URL, and File record types in the F00165 table based on object name (OBNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllHTMLKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllHTML(_J("ABGT") , &dsABGT);

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllImage(
        PJSTR szObjectName,
        LPVOID lpMODSKey
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllImageKeyStr(
        PJSTR szObjectName,
        PJSTR pszMOKeyStr
        );
```

Description

This function deletes all image record types in the F00165 table based on object name (BNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples demonstrating how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllImageKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllImage(_J("ABGT") , &dsABGT);

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllMOType(
          PJSTR szObjectName,
          LPVOID lpMODSKey
          );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllMOTypeKeyStr(
          PJSTR szObjectName,
          PJSTR pszMOKeyStr
          );
```


Description

This function deletes all media object record types in the F00165 table based on object name (OBNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllMOTypeKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllMOType (_J("ABGT") , &dsABGT);

return;
```


See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllOLE(
        PJSTR szObjectName,
        LPVOID lpMODSKey
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllOLEKeyStr(
        PJSTR szObjectName,
        PJSTR pszMOKeyStr
        );
```


Description

This function deletes all OLE record types in the F00165 table based on object name (BNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllOLEKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllOLE(_J("ABGT") , &dsABGT);

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllShortcut(
          PJSTR szObjectName,
          LPVOID lpMODSKey
          );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllShortcutKeyStr(
          PJSTR szObjectName,
          PJSTR pszMOKeyStr
          );
```


Description

This function deletes all shortcut record types in the F00165 table based on object name (OBNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllShortcutKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllShortcut (_J("ABGT") , &dsABGT);

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllText(
        PJSTR szObjectName,
        LPVOID lpMODSKey
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllTextKeyStr(
        PJSTR szObjectName,
        PJSTR pszMOKeyStr
        );
```

Description

This function deletes all text record types in the F00165 table based on object name (OBNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.

Define Type	Note
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllTextKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllText(_J("ABGT") , &dsABGT);

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllVendor(
        PJSTR szObjectName,
        LPVOID lpMODSKey
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_AllVendorKeyStr(
        PJSTR szObjectName,
        PJSTR pszMOKeyStr
```



```
);
```

Description

This function deletes all vendor or third-party software record types in the F00165 table based on object name (OBNM) and object keys (TXKY).

Parameters

Parameter	Description
szObjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY.
pszMOKeyStr	Input, required. Formatted string used for TXKY.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.

Define Type	Note
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;

JDBReturn = jdeGTDelete_AllVendorKeyStr(_J("ABGT") , _J("1"));

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTDelete_AllVendor _J("ABGT") , &dsABGT);

return;
```


See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_HTML/jdeGTDelete_HTMLKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_HTML(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_HTMLKeyStr(
         PJSTR szObjectName,
```



```
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function deletes specific HTML, URL and file record types from the F00165 table based on the *pMOData* parameter.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_HTMLKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
```

```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_HTML(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_Image/jdeGTDelete_ImageKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_Image(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_ImageKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function deletes specific image record types from the F00165 table based on the *pMOData* parameter.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others

Data Type	Data Description	Note
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_ImageKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}
```



```
return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Image(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_OLE/jdeGTDelete_OLEKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_OLE(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_OLEKeyStr(
```



```
PJSTR szObjectName,
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function deletes specific OLE record types from the F00165 table based on *pMOData*.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
ITotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_OLEKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
```



```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_OLE(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_Shortcut/jdeGTDelete_ShortcutKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_Shortcut(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_ShortcutKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function deletes specific shortcut record types from the F00165 table based on the *pMOData* parameter.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others

Data Type	Data Description	Note
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_ShortcutKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}
```

```
return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Shortcut (_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_Text/jdeGTDelete_TextKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_Text(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        LPMODATA pMOData,
        long lTotalRec
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_TextKeyStr(
```



```
PJSTR szObjectName,
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function deletes a specific record type from the F00165 table based on *pMOData*.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
ITotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_TextKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
```

```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Text(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTDelete_Vendor/jdeGTDelete_VendorKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_Vendor(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTDelete_VendorKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function deletes specific vendors or third-party software record types from the F00165 table based on the *pMOData* parameter.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.

Data Type	Data Description	Note
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
```

```
    jdeGTDelete_VendorKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
    jdeFreeMOData(pMOData, lTotalRec)
  }

  return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Vendor(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTFreeMOData

Syntax

```
JDERTN(void) JDEWINAPI jdeGTFreeMOData(
        LPMODATA lpMOData,
        long lNumOfRec
        );
```


Description

This function frees the memory pointer of the allocated array of the MODATA structure. It also frees any memory pointer of the *pData* member in MODATA.

Parameters

Parameter	Description
lpMOData	Input, required. Allocate memory of the array of pointer to MODATA structure. The memory is freed along with any *pData* pointer.
INumOfRec	Input, required. The number of elements in the array pointer.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.

Define Type	Note
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_TextKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Text(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_AllMOType(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        LPMODATA pMOData,
        long lTotalRec
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_AllMOTypeKeyStr(
```



```
PJSTR szObjectName,
PJSTR pszMOKeyStr,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function retrieves all record types from the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_AllMOTypeKeyStr (_J("ABGT"), _J("1"), &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_AllMOType(_J("ABGT"), &dsABGT, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
```



```
{
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_GenericText(
        PJSTR szObjectName,
```



```
          LPVOID lpMODSKey,
          int nSeq,
          LPMODATA pMOData,
          long lTotalRec
          );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_GenericTextKeyStr(
          PJSTR szObjectName,
          PJSTR pszMOKeyStr,
          int nSeq,
          LPMODATA pMOData,
          long lTotalRec
          );
```

Description

This function retrieves the text record type from the F00165 table and converts the retrieved RTF text to plain text.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name

Data Type	Data Description	Note
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_GenericTextKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotal⇒
Rec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_TextKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}
```



```
return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_GenericText(_J("ABGT"), &dsABGT, 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Text(_J("ABGT") , &dsABGT, pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_HTML/jdeGTGet_HTMLKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_HTML(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        int nSeq,
        LPMODATA pMOData,
        long lTotalRec
        );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_HTMLKeyStr(
```



```
PJSTR szObjectName,
PJSTR pszMOKeyStr,
int nSeq,
LPMODATA pMOData,
long lTotalRec
);
```

Description

This function retrieves the HTML, URL, and file record type from the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_HTMLKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_HTMLKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;

JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_HTML(_J("ABGT"), &dsABGT, 0,&pMOData, &lTotalRec);
```



```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_HTML(_J("ABGT") , &dsABGT, pMOData, lTotalRec)
  jdeFreeMOData(pMOData, lTotalRec)
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_Image/jdeGTGet_ImageKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_Image(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_ImageKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function retrieves the image record type from the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.

Data Type	Data Description	Note
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_ImageKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
```



```
  jdeGTDelete_ImageKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_Image(_J("ABGT"), &dsABGT, 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Image(_J("ABGT") , &dsABGT, pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_OLE/jdeGTGet_OLEKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_OLE(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        int nSeq,
        LPMODATA pMOData,
        long lTotalRec
        );
```

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_OLEKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function retrieves the OLE record type from the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_OLEKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_OLEKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));
```

```
JDBReturn = jdeGTGet_OLE(_J("ABGT"), &dsABGT, 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_OLE(_J("ABGT") , &dsABGT, pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_RTFText(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_RTFTextKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function retrieves the text record type from the F00165 table and does not affect the RTF text.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.

Parameter	Description
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.

Data Type	Data Description	Note
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_RTFTextKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotalRec);
```



```
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_TextKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_RTFText(_J("ABGT"), &dsABGT, 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Text(_J("ABGT") , &dsABGT, pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_Shortcut(
         PJSTR szObjectName,
         LPVOID lpMODSKey,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
```



```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_ShortcutKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function retrieves the shortcut record type from the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.

Parameter	Description
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
ITotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	

Data Type	Data Description	Note
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_ShortcutKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_ShortcutKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```


This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));

JDBReturn = jdeGTGet_Shortcut(_J("ABGT"), &dsABGT, 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Shortcut(_J("ABGT") , &dsABGT, pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGet_Vendor/jdeGTGet_VendorKeyStr

Syntax

```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_Vendor(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        int nSeq,
        LPMODATA pMOData,
        long lTotalRec
        );
```



```
JDERTN(JDEDB_RESULT) JDEWINAPI jdeGTGet_VendorKeyStr(
         PJSTR szObjectName,
         PJSTR pszMOKeyStr,
         int nSeq,
         LPMODATA pMOData,
         long lTotalRec
         );
```

Description

This function retrieves the vendor record type from the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOData	Input, required. Array of data structure that stores the data to be deleted.
lTotalRec	Input, required. Indicates the number of array elements in *pMOData*.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;

JDBReturn = jdeGTGet_VendorKeyStr (_J("ABGT"), _J("1"), 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_VendorKeyStr(_J("ABGT") , _J("1"), pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
LPMODATA  pMOData = NULL;
long lTotalRec = 0;
DSABGT dsABGT = {0};

ParseNumericString(&dsABGT.mnAddressNumber, _J("1"));
```



```
JDBReturn = jdeGTGet_Vendor(_J("ABGT"), &dsABGT, 0, &pMOData, &lTotalRec);
if (JDBReturn == JDEDB_PASSED && pMOData && lTotalRec > 0)
{
  jdeGTDelete_Vendor(_J("ABGT") , &dsABGT, pMOData, lTotalRec);
  jdeFreeMOData(pMOData, lTotalRec);
}

return;
```

See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGetCount/jdeGTGetCountKeyStr, page 284
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeGTGetCount/jdeGTGetCountKeyStr

Syntax

```
JDERTN(long) JDEWINAPI jdeGTGetCount(
       PJSTR szObjectName,
       LPVOID lpMODSKey,
       MOTYPE nMOType
       );
JDERTN(long) JDEWINAPI jdeGTGetCountKeyStr(
       PJSTR szObjectName,
       PJSTR pszMOKeyStr,
       MOTYPE nMOType
       );
```

Description

This function retrieves the number of media object records the F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOType	Input, required. Media object type.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	

Data Type	Data Description	Note
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
>0	Number of records found.
≤	Either no records exist or the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
long lTotalRecFound = 0;
BOOL bSuccess = TRUE;

lTotalRecFound = jdeValidateGTExist(_J("ABGT") , &dsABGT, OBJ_RTFTEXT);
if (lTotalRecFound > 0)
{
 bSuccess = TRUE;
}
else
{
 bSuccess = FALSE;
}
```

```
return;
```

This is the second example of how to use this API:

```
long lTotalRecFound = 0;
DSABGT dsABGT = {0};
BOOL bSuccess = TRUE;

ParseNumericString(&dsABGT.mnAddressNumber, J("1"));

lTotalRecFound = jdeValidateGTExist(_J("ABGT") , &dsABGT, OBJ_RTFTEXT);
if (lTotalRecFound > 0)
{
 bSuccess = TRUE;
}
else
{
 bSuccess = FALSE;
}

return;
```


See Also

Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTFreeMOData, page 252
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, "JD Edwards EnterpriseOne APIs," jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, "JD Edwards EnterpriseOne APIs," jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 287

jdeValidateGTExist/jdeValidateGTExistWithKeyStr

Syntax

```
JDERTN(long) JDEWINAPI jdeValidateGTExist(
        PJSTR szObjectName,
        LPVOID lpMODSKey,
        MOTYPE nMOType
        );
JDERTN(long) JDEWINAPI jdeValidateGTExistWithKeyStr(
        PJSTR szObjectName,
```



```
PJSTR pszMOKeyStr,
MOTYPE nMOType
);
```

Description

This function checks whether any media object records exist in F00165 table.

Parameters

Parameter	Description
szOjectName	Input, required. GT data structure name. Primary unique key.
lpMODSKey	Input, required. GT data structure with valid data. The data within the GT data structure is formatted into a string used for TXKY
pszMOKeyStr	Input, required. Formatted string used for TXKY.
pMOType	Input, required. Media object type.

Additional Notes

This table describes the MODATA (or *LPMODATA) data structure definition:

Data Type	Data Description	Note
int	nSeq	Sequence number form MOSEQN.
MOTYPE	nMOType	Media object type.
JCHAR	szUser[11]	User name.
JDEDATE	jdDate	Date updated.
MATH_NUMERIC	mnTime	Time updated.
BOOL	bRTFData	TRUE = RTF Text FALSE = Plain Text or others
JCHAR	szItemName[GT_ITNMSIZE]	Item name
JCHAR	szQueueName[GT_QUESIZE]	
JCHAR	szFileName[GT_FILESIZE]	
PJSTR	pData	Allocate memory for text and shortcut media object type.

This table describes the MOTYPE definition:

Define Type	Note
OBJ_JDEALL	All media object types
OBJ_RTFTEXT	Text media object.
OBJ_JDEIMAGE	Image media object.
OBJ_JDEOLE	OLE media object.
OBJ_MISCJDESHORTCUT	Shortcut media object.
OBJ_MISIMAGEVENDOR	Third-party vendor.
OBJ_MISCHTML	HTML/URL/File media object.

Returns

This API can return these values:

Value	Description
JDEDB_PASSED	Indicates that the API succeeded.
JDEDB_FAILED	Indicates that the API failed.

Example

This is the first of two examples that demonstrate how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
BOOL bSuccess = TRUE;

JDBReturn = jdeValidateGTExistWithKeyStr (_J("ABGT") , J("1"), OBJ_JDEALL);
if (JDBReturn == JDEDB_PASSED)
{
 bSuccess = TRUE;
}
else
{
 bSuccess = FALSE;
}

return;
```

This is the second example of how to use this API:

```
JDEDB_RESULT JDBReturn = JDEDB_PASSED;
DSABGT dsABGT = {0};
BOOL bSuccess = TRUE;

ParseNumericString(&dsABGT.mnAddressNumber, J("1"));
```

```
JDBReturn = jdeValidateGTExist(_J("ABGT") , &dsABGT, OBJ_JDEALL);
if (JDBReturn == JDEDB_PASSED)
{
 bSuccess = TRUE;
}
else
{
 bSuccess = FALSE;
}

return;
```

See Also

Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 186
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 189
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 193
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 196
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 200
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 203
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 207
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 210
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 213
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 216
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 219
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 222
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 225
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 228
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTFreeMOData, page 252
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 255
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 258
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 262
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 266
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 269
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 273
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 276
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 280
Appendix A, “JD Edwards EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 284

Messaging and Workflow APIs

This section discusses APIs for systems that use messaging and workflow such as WorkCenter.

DoSendMessagev3

Syntax

```
MSGPRTO_RTN(JDEDB_RESULT) DoSendMessageV3(
        HUSER hUser,
        Recipient * to,
        Recipient * cc,
        Recipient * bcc,
        const JCHAR * pSubject,
        const JCHAR * pText,
        int numActiveMsg,
        const ACTIVE_MSG_INFO_V3 * pActiveMsgArray,
        const MSG_TEMPLATE_INFO * pTemplateSub,
        const JCHAR * mediaObjectName,
        const JCHAR * mediaObjectKey,
        const JCHAR * pMailBox
        );
```

Description

This API sends an email, internal or external, based on the recipient's preference.

Parameters

Parameter	Description
HUSER *hUser*	Input, required. The user handle that defines the context in which to send the message.
Recipient * *to*	Input, required. The ID of the primary individual, group, mailbox, and so forth to whom the message is to be delivered.
Recipient * *cc*	Input, required. The ID of the individual, group, mailbox, and so forth to whom a courtesy copy of the message is to be delivered.
Recipient * *bcc*	Input, required. The ID of the individual, group, mailbox, and so forth to whom a blind courtesy copy of the message is to be delivered.
const JCHAR * *pSubject*	Input, optional. The text to write to the subject line for the email.
const JCHAR * *pText*	Input, optional. The text to write to the main body for the email.
int *numActiveMsg*	Input, required. The number of interconnects to specific forms required.
const ACTIVE_MSG_INFO_V3 * *pActiveMsgArray*	Input, optional. Array holding the shortcuts. NULL is a value.
const MSG_TEMPLATE_INFO * *pTemplateSub*	Input, optional. A specific text substitution template.
const JCHAR * *mediaObjectName*	Input, optional. A specific media object used to send the attachment. NULL and <blank> are values.
const JCHAR * *mediaObjectKey*	Input, optional. The key of the media object used to send the attachment.
const JCHAR * *pMailBox*	Input, optional. The internal WorkCenter mailbox to which to deliver the email (used only for PPAT).

Example

```
JDEBFRTN (ID) JDEBFWINAPI functionCRMTest1 (LPBHVRCOM lpBhvrCom, LPVOID lpVoid,⇒
 LPDSDCRM1 lpDS)

{
 JCHAR buffer[1000] = {0};
 JDEDB_RESULT rc = JDEDB_PASSED;
 Recipient to = {0};
 Recipient cc = {0};
 HUSER hUser = NULL;
 MSG_TEMPLATE_INFO zTemplateInfo = { 0 };

 JDB_InitBhvr(lpBhvrCom, &hUser, (JCHAR *)NULL, JDEDB_COMMIT_AUTO);

 jdeUTime_Format(buffer, (LPJDEUTIME)&lpDS->DateEntered, NULL);

 /*Create TO smtp recipient*/
```

```
 to.recipientType = RECIPIENT_TYPE_SMTP;
 to.smtp = _J("employee@oracle.com");

 /*Create CC contact recipient*/
 cc.recipientType = RECIPIENT_TYPE_CONTACT;
 LongToMathNumeric(1001, &cc.an8);
 LongToMathNumeric(0, &cc.idln);

 /*Create a template sub with dditem LM1234*/
 zTemplateInfo.ddName = (JCHAR*)jdeAlloc( COMMON_POOL , (sizeof(NID)) * sizeof⇒
(JCHAR) ,MEM_ZEROINIT );
 jdeNIDcpy( zTemplateInfo.ddName , _J("LM1234") );
 zTemplateInfo.nbParam = 6;
 zTemplateInfo.valueArray = (RT_VALUE*) jdeAlloc(COMMON_POOL, (sizeof(RT_VALUE)⇒
 * 6) , MEM_ZEROINIT | MEM_RESIZEABLE);
 zTemplateInfo.valueArray[0].evdtType = 2;
 zTemplateInfo.valueArray[0].value = jdemalloc(sizeof(JCHAR) * (jdeStrlen(_J⇒
("1")) + 1));
 jdeStrncpyTerminate(zTemplateInfo.valueArray[0].value , (LPVOID)_J("1") , jde⇒
Strlen(_J("1")) + 1 );
 zTemplateInfo.valueArray[1].evdtType = 2;
 zTemplateInfo.valueArray[1].value = jdemalloc(sizeof(JCHAR) * (jdeStrlen(_J⇒
("2")) + 1));
 jdeStrncpyTerminate(zTemplateInfo.valueArray[1].value , (LPVOID)_J("2") , jde⇒
Strlen(_J("2")) + 1 );
 zTemplateInfo.valueArray[2].evdtType = 2;
 zTemplateInfo.valueArray[2].value = jdemalloc(sizeof(JCHAR) * (jdeStrlen(_J⇒
("3")) + 1));
 jdeStrncpyTerminate(zTemplateInfo.valueArray[2].value , (LPVOID)_J("3") , jde⇒
Strlen(_J("3")) + 1 );
 zTemplateInfo.valueArray[3].evdtType = 2;
 zTemplateInfo.valueArray[3].value = jdemalloc(sizeof(JCHAR) * (jdeStrlen(_J⇒
("4")) + 1));
 jdeStrncpyTerminate(zTemplateInfo.valueArray[3].value , (LPVOID)_J("4") , jde⇒
Strlen(_J("4")) + 1 );
 zTemplateInfo.valueArray[4].evdtType = 2;
 zTemplateInfo.valueArray[4].value = jdemalloc(sizeof(JCHAR) * (jdeStrlen(_J⇒
("5")) + 1));
 jdeStrncpyTerminate(zTemplateInfo.valueArray[4].value , (LPVOID)_J("5") , jde⇒
Strlen(_J("5")) + 1 );
 zTemplateInfo.valueArray[5].evdtType = 2;
 zTemplateInfo.valueArray[5].value = jdemalloc(sizeof(JCHAR) *
   (jdeStrlen(_J("6")) + 1));
 jdeStrncpyTerminate(zTemplateInfo.valueArray[5].value , (LPVOID)_J("6") , jde⇒
Strlen(_J("6")) + 1 );

 rc = DoSendMessageV3(
  hUser,   /*HUSER*/
  &to,   /*to*/
  &cc,   /*cc*/
```



```
      NULL,   /*bcc*/
      buffer,  /*subject*/
      buffer,  /*messageText*/
      0,    /*numActiveMsg number of shortcuts*/
      NULL,   /*pActiveMsgArray optional, NULL allowed */
      &zTemplateInfo,/*pTemplateSub optional, NULL allowed */
      NULL,   /*mediaObjectName optional, NULL or <blank> allowed */
      NULL,   /*mediaObjectKey optional, NULL or <blank> allowed */
      NULL);   /*pMailBox optional, only used for PPAT */

     FreeActiveMsgTemplate(&zTemplateInfo);
     return (ER_SUCCESS);
    }
```

SAX Interface Functions

This section describes the SAX Parser interface functions.

Structure Used With SAX Parser Interface Functions

Syntax

This structure is used to pass attribute data to the start-element callback functions:

```
typedef struct tagXRCS_Attr_Info XRCS_ATTR_INFO;
struct tagXRCS_Attr_Info {
 const JCHAR *szAttrLocalname;
 const JCHAR *szAttrQname;
 const JCHAR *szAttrValue;
};
```

XRCS_initEngine

Syntax

```
XRCS_Status XRCS_initEngine (void);
```

Description

This function performs a one-time initialization of the XercesWrapper and the Xerces parsing code. It must be the first XercesWrapper function called from the client code.

XRCS_getParserByType

Syntax

```
XRCS_getParserByType(
      XRCS_hParser* phParser,
      XRCS_ParserType eParserType
      );
```


Description

This function initializes a SAX or DOM parser, and returns a handle to it.

Parameters

Parameter	Description
phParser	A valid pointer to a parser handle (address IN, handle OUT).
eParserType	A type of parser (DOM or SAX), specified by enum value (IN).

XRCS_getParser (DOM only)

Syntax

```
XRCS_Status XRCS_getParser(XRCS_hParser* phParser);
```

Description

This function initializes a DOM parser, and returns a handle to it. Do not use this function for the SAX parser. Instead, use **XRCS_getParserByType** with the type set to `XRCS_SAX_PARSER_TYPE`.

Parameters

Parameter	Description
phParser	A valid pointer to a parser handle (address IN, handle OUT).

XRCS_setCallback

Syntax

```
XRCS_Status XRCS_setCallback(       const XRCS_hParser hParser
,       XRCS_CallbackType eCallbackType
,       void *pCallbackFunction
,       void *pContext
       );
```

Description

This function sets up the given callback function for the given SAX parsing event. The function prototype of the callback function must correspond to the type of callback. The callback function must be cast to (void *). The context pointer must be cast to (void *). The context pointer typically points to a user-defined data structure, where the callback functions can maintain the state of the parsing. The context pointer may be set to NULL if it is not needed.

You can set up multiple callback functions for the same parsing event type. However, the order in which they are called cannot be specified.

Parameters

Parameter	Description
hParser	Input. A valid parser handle.
eCallbackType	Input. A type of callback, specified by enum value.
pCallbackFunction	Input. A pointer to callback function.
pContext	A pointer to be passed to the callback function.

XRCS_setCallbackWithOption

Syntax

```
XRCS_Status XRCS_setCallbackWithOption(
        const XRCS_hParser hParser,
        XRCS_CallbackType eCallbackType,
        void * pCallbackFunction,
        void * pContext,
        XRCS_CallbackOptionType eOptionType,
        void * OptionValue
        );
```

Description

This function is the same as **XRCS_setCallback**, with the ability to specify an option. The type of option that may be specified is based upon the type of parsing event. Some types of options require a value, in which case the value should always be cast to (void *). If no value is required, a NULL can be used. The system currently supports two types of options.

For the start-of-element and end-of-element event types, an optional, specified local name may be passed in. This option type is identified by the enum value, *XRCS_CBOPT_ELEM_LOCAL_NAME*. The option value would be a null-terminated string (JCHAR *) for the element name. The callback function will be triggered only when the local name for the element matches the passed-in name.

For the characters event type, you have the option to retrieve the text following the end of an element. The option type is identified by the enum value, *XRCS_CBOPT_CHARS_AFTER_ELEM*. The option value is ignored, so you should set it to NULL. For typical XML data, the characters after the end of an element are just white space and carriage returns and so do not provide any useful information. On the other hand, XML display documents could have useful display information after the end of an element. The default is to ignore any text following the end of an element.

Parameters

Parameter	Description
hParser	Input. A valid parser handle.
eCallbackType	Input. A type of callback, specified by enum value.
pCallbackFunction	Input. A pointer to callback function.
pContext	Input. A pointer to be passed to the callback function.
eOptionType	Input. A type of option, specified by enum value.
pOptionValue	Input. A value for the option. Not all options require a value.

XRCS_parseXMLFile

Syntax

```
XRCS_Status JDEWINAPI XRCS_parseXMLFile(
        const XRCS_hParser hParser,
        const JCHAR* szFileName,
        XRCS_hDocument* phDoc
        );
```

Description

This function parses the given XML file. It is used for both SAX and DOM parsers. For the SAX parser, the document handle pointer is not used, so it must be passed in as NULL. Certain error conditions could cause a process crash if the document handle pointer is not set to NULL for the SAX parser.

The SAX parser will stop parsing the XML text whenever:

- The end of the XML text has been reached.
- A callback has requested to terminate parsing (using its return code).
- The SAX parser encounters a fatal error.

In the first case, the parse function returns XRCS_SUCCESS. In the last two, the parse function returns XRCS_ERROR.

After the callback functions have been set up, the XML parse functions may be called multiple times. Multiple calls might be useful for parsing multiple XML files while using the same group of callback functions.

Parameters

Parameter	Description
hParser	Input. A valid parser handle.
szFileName	Input. The XML data file to be parsed.
phDoc	Input. A pointer to a DOM document handle. For SAX, pass in NULL.

XRCS_parseXMLString

Syntax

```
XRCS_Status JDEWINAPI XRCS_parseXMLString(
        XRCS_hParser hParser,
        const JCHAR* szXMLString,
        XRCS_hDocument* phDoc
        );
```

Description

This function is the same as **XRCS_parseXMLFile**, with the only difference being the source of the XML text.

Parameters

Parameter	Description
hParser	Input. A valid parser handle.
szXMLString	Input. An XML text string to be parsed.
phDoc	Input. A pointer to a DOM document handle. For SAX, pass in NULL

XRCS_freeParser

Syntax

XRCS_Status XRCS_freeParser(XRCS_hParser *hParser*);

Description

This function frees the resources that are used by the parser.

Parameters

Parameter	Description
hParser	Input. A valid parser handle.

XRCS_terminateEngine

Syntax

```
XRCS_Status XRCS_terminateEngine(void);
```

Description

This function performs a one-time cleanup of resources used by XercesWrapper and Xerces parsers. It must be the last function called from the client code.

Callback Functions

Because the available data is different for each event type, the callback functions have specified parameter lists (function prototypes), based upon the associated type of SAX parsing event. All callback functions receive the context pointer, which was set up before parsing begins. Additional parameters correspond to the type of event.

All values passed into the callback functions must be considered temporary. If the data needs to be kept, it should be copied elsewhere. Data should never be saved using a reference pointer to the passed-in data. Also, the passed-in data must never be modified. That memory, used to pass data into the callback functions, will either be freed or reused upon return from the callback function.

All callback functions return an enum value, which indicates whether the parsing should continue or terminate. Whenever a callback function requests a termination, the XML parsing will stop, and the parse function itself will return an error code.

The context pointer must always be cast to (void *). The callback function must always be cast to (void *). Without the (void *) casts, the code will work on most systems. However, it will not build on Linux systems without the explicit casts.

Errors and Warnings

Description

After a parser fatal error, the parse function returns an error status, regardless of whether the callback functions return "continue" or "terminate." When the SAX parser encounters a fatal error, it first calls the fatal error callbacks, and then stops parsing.

If multiple fatal-error callbacks are set up, returning the "terminate" return code will skip the remaining callbacks and stop parsing immediately. If multiple fatal-error callbacks are set up, returning the "continue" return code will enable the remaining callbacks to run, and then stop parsing. If only one fatal-error callback is set up, then the return code does not matter because parsing will always stop after that callback finishes.

The SAX parser interface project did not change the threading capabilities of the XercesWrapper code. It uses the same initialization code as the DOM parser. Both parsers are not completely threadsafe, particularly for initialization and termination. The calling functions are responsible for all thread control.

If the MS Windows VC++ build uses the /W4 switch (warning level 4), the system will issue compiler warnings for the callback function casting, in the calls to **XRCS_setCallback** and **XRCS_setCallbackWithOption**. The /W4 switch appears to be the default for business function builds. To eliminate the warnings, add this block of code before the first call to **XRCS_setCallback** or **XRCS_setCallbackWithOption**. The block of code can be placed in a header file, if multiple files setting up callbacks exist:

```
#ifdef JDENV_PC
/* Do not display warning for callback functions -- ignore warning
* for conversion of function pointer to data pointer (void *).
* Compiler normally displays this warning when using /W4 warnings level.
*/
#pragma warning (disable:4054)
#endif
```


Callback Function Format 1

Syntax

```
XRCS_CallbackStatus ( * PCALLBACK_FORMAT1) ( void *pContext);
```

Description

This is the function prototype of callback functions, which is used by event types for start and end of documents.

Parameters

Parameter	Description
pContext	Input. A pointer to the context, which was specified during setup.

Callback Function Format 2

Syntax

```
XRCS_CallbackStatus ( * PCALLBACK_FORMAT2) (
            void * pContext,
            const JCHAR * szUri,
            const JCHAR * szLocalname,
            const JCHAR * szQname,
            unsigned int nNumAttrs,
            const XRCS_ATTR_INFO * pAttributes
            );
```

Description

This is the function prototype of callback functions, which is used by the event type for start of an element. The attribute information is returned in an array. If no attribute information exists, the number of elements is set to zero, and the array pointer is set to NULL.

Each element of the attribute array consists of the data structure **XRCS_ATTR_INFO**. That structure contains three null-terminated strings: one for the local name of the attribute, one for the qualified name of the attribute, and one for the value of the attribute.

Parameters

Parameter	Description
pContext	Input. A pointer to the context, which was specified during setup.
szUri	Input. A null-terminated string for URI (namespace).
szLocalname	Input. A null-terminated string for local name.
szQname	Input. A null-terminated string for qualified name.
nNumAttrs	Input. The number of elements in the array of attribute information.
pAttributes	Input. An array of attribute information.

Callback Function Format 3

Syntax

```
XRCS_CallbackStatus ( * PCALLBACK_FORMAT3) (
            void * pContext,
            const JCHAR * szUri,
            const JCHAR * szLocalname,
            const JCHAR * szQname
            );
```

Description

This is the function prototype of callback functions, which is used by the event type for end of element.

Parameters

Parameter	Description
pContext	Input. A pointer to the context, which was specified during setup.
szUri	Input. A null-terminated string for URI (namespace).
szLocalname	Input. A null-terminated string for local name.
szQname	Input. A null-terminated string for qualified name.

Callback Function Format 4

Syntax

```
XRCS_CallbackStatus ( * PCALLBACK_FORMAT4) (
            void * pContext,
            const JCHAR * szText
            );
```

Description

This is the function prototype of callback functions, which is used by event types for characters and ignorable white space. For the event type of the characters, the complete character string is returned, even if the SAX parser returns it to the XercesWrapper code as a series of partial strings.

The event for ignorable white space occurs only when using schemas and other special features. This version of the SAX parser interface does not include a method to set up schemas. Therefore, any callback functions which are set up for ignorable white space will never be called. After schema-setup is added in a later version, the ignorable white space callbacks will be called. Without schemas, all text is returned using the event type for the characters.

Parameters

Parameter	Description
pContext	Input. A pointer to the context, which was specified during setup.
szText	Input. A null-terminated character string.

Callback Function Format 5

Syntax

```
XRCS_CallbackStatus ( * PCALLBACK_FORMAT5) (
            void * pContext,
            XRCS_CallbackType eCallbackType,
            int nLineNum,
            int nColNum,
            const JCHAR * szPublicId,
            const JCHAR * szSystemId,
            const JCHAR * szMessage
            );
```

Description

This is the function prototype of callback functions, which is used by event types for warnings, errors, and fatal errors. The enum for callback type makes it possible for one callback function to handle all three error types. The line and column numbers are approximate, and may point to the position following the actual error. That is because the SAX parser may have already moved its pointers to the next element, before an error is encountered with the current element.

The XML file name is usually found in either *szPublicId* or *szSystemId*. The other one is usually an empty (zero-length) string. The error message text (*szMessage*) is sometimes an empty (zero-length) string.

Parameters

Parameter	Description
pContext	Input. A pointer to the context, which was specified during setup.
eCallbackType	Input. An enum indicating type of error event.
nLineNum	Input. A line number where error occurred.
nColNum	Input. The column number where error occurred.
szPublicId	Input. The null-terminated name of XML file.
szSystemId	Input. The null-terminated name of XML file.
szMessage	Input. The null-terminated text of error message.

Glossary of JD Edwards EnterpriseOne Terms

activity	A scheduling entity in JD Edwards EnterpriseOne tools that represents a designated amount of time on a calendar.
activity rule	The criteria by which an object progresses from one given point to the next in a flow.
add mode	A condition of a form that enables users to input data.
Advanced Planning Agent (APAg)	A JD Edwards EnterpriseOne tool that can be used to extract, transform, and load enterprise data. APAg supports access to data sources in the form of rational databases, flat file format, and other data or message encoding, such as XML.
application server	A server in a local area network that contains applications shared by network clients.
as if processing	A process that enables you to view currency amounts as if they were entered in a currency different from the domestic and foreign currency of the transaction.
alternate currency	A currency that is different from the domestic currency (when dealing with a domestic-only transaction) or the domestic and foreign currency of a transaction. In JD Edwards EnterpriseOne Financial Management, alternate currency processing enables you to enter receipts and payments in a currency other than the one in which they were issued.
as of processing	A process that is run as of a specific point in time to summarize transactions up to that date. For example, you can run various JD Edwards EnterpriseOne reports as of a specific date to determine balances and amounts of accounts, units, and so on as of that date.
back-to-back process	A process in JD Edwards EnterpriseOne Supply Management that contains the same keys that are used in another process.
batch processing	A process of transferring records from a third-party system to JD Edwards EnterpriseOne. In JD Edwards EnterpriseOne Financial Management, batch processing enables you to transfer invoices and vouchers that are entered in a system other than JD Edwards EnterpriseOne to JD Edwards EnterpriseOne Accounts Receivable and JD Edwards EnterpriseOne Accounts Payable, respectively. In addition, you can transfer address book information, including customer and supplier records, to JD Edwards EnterpriseOne.
batch server	A server that is designated for running batch processing requests. A batch server typically does not contain a database nor does it run interactive applications.
batch-of-one immediate	A transaction method that enables a client application to perform work on a client workstation, then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks. See also direct connect and store-and-forward.
business function	A named set of user-created, reusable business rules and logs that can be called through event rules. Business functions can run a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the application programming interfaces (APIs) that enable them to be called from a form, a database trigger, or a non-JD Edwards EnterpriseOne 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 (NER).

business view A means for selecting specific columns from one or more JD Edwards EnterpriseOne application tables whose data is used in an application or report. A business view does not select specific rows, nor does it contain any actual data. It is strictly a view through which you can manipulate data.

central objects merge A process that blends a customer's modifications to the objects in a current release with objects in a new release.

central server A server that has been designated to contain the originally installed version of the software (central objects) for deployment to client computers. In a typical JD Edwards EnterpriseOne installation, the software is loaded on to one machine—the central server. Then, copies of the software are pushed out or downloaded to various workstations attached to it. That way, if the software is altered or corrupted through its use on workstations, an original set of objects (central objects) is always available on the central server.

charts Tables of information in JD Edwards EnterpriseOne that appear on forms in the software.

connector Component-based interoperability model that enables third-party applications and JD Edwards EnterpriseOne to share logic and data. The JD Edwards EnterpriseOne connector architecture includes Java and COM connectors.

contra/clearing account A general ledger account in JD Edwards EnterpriseOne Financial Management that is used by the system to offset (balance) journal entries. For example, you can use a contra/clearing account to balance the entries created by allocations in JD Edwards EnterpriseOne Financial Management.

Control Table Workbench An application that, during the Installation Workbench processing, runs the batch applications for the planned merges that update the data dictionary, user-defined codes, menus, and user override tables.

control tables merge A process that blends a customer's modifications to the control tables with the data that accompanies a new release.

cost assignment The process in JD Edwards EnterpriseOne Advanced Cost Accounting of tracing or allocating resources to activities or cost objects.

cost component In JD Edwards EnterpriseOne Manufacturing, an element of an item's cost (for example, material, labor, or overhead).

cross segment edit A logic statement that establishes the relationship between configured item segments. Cross segment edits are used to prevent ordering of configurations that cannot be produced.

currency restatement The process of converting amounts from one currency into another currency, generally for reporting purposes. You can use the currency restatement process, for example, when many currencies must be restated into a single currency for consolidated reporting.

database server A server in a local area network that maintains a database and performs searches for client computers.

Data Source Workbench An application that, during the Installation Workbench process, copies all data sources that are defined in the installation plan from the Data Source Master and Table and Data Source Sizing tables in the Planner data source to the system-release number data source. It also updates the Data Source Plan detail record to reflect completion.

date pattern	A calendar that represents the beginning date for the fiscal year and the ending date for each period in that year in standard and 52-period accounting.
denominated-in currency	The company currency in which financial reports are based.
deployment server	A server that is used to install, maintain, and distribute software to one or more enterprise servers and client workstations.
detail information	Information that relates to individual lines in JD Edwards EnterpriseOne transactions (for example, voucher pay items and sales order detail lines).
direct connect	A transaction method in which a client application communicates interactively and directly with a server application. See also batch-of-one immediate and store-and-forward.
Do Not Translate (DNT)	A type of data source that must exist on the iSeries because of BLOB restrictions.
dual pricing	The process of providing prices for goods and services in two currencies.
edit code	A code that indicates how a specific value for a report or a form should appear or be formatted. The default edit codes that pertain to reporting require particular attention because they account for a substantial amount of information.
edit mode	A condition of a form that enables users to change data.
edit rule	A method used for formatting and validating user entries against a predefined rule or set of rules.
Electronic Data Interchange (EDI)	An interoperability model that enables paperless computer-to-computer exchange of business transactions between JD Edwards EnterpriseOne and third-party systems. Companies that use EDI must have translator software to convert data from the EDI standard format to the formats of their computer systems.
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 the business function event rule.
Employee Work Center	A central location for sending and receiving all JD Edwards EnterpriseOne 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.
enterprise server	A server that contains the database and the logic for JD Edwards EnterpriseOne.
EnterpriseOne 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.
EnterpriseOne process	A software process that enables JD Edwards EnterpriseOne clients and servers to handle processing requests and run transactions. A client runs one process, and servers can have multiple instances of a process. JD Edwards EnterpriseOne 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.
Environment Workbench	An application that, during the Installation Workbench process, copies the environment information and Object Configuration Manager tables for each environment from the Planner data source to the system-release number data source. It also updates the Environment Plan detail record to reflect completion.
escalation monitor	A batch process that monitors pending requests or activities and restarts or forwards them to the next step or user after they have been inactive for a specified amount of time.

event rule	A logic statement that instructs the system to perform one or more operations based on an activity that can occur in a specific application, such as entering a form or exiting a field.
facility	An 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. A facility is sometimes referred to as a "business unit."
fast path	A command prompt that enables the user to move quickly among menus and applications by using specific commands.
file server	A server that stores files to be accessed by other computers on the network. Unlike a disk server, which appears to the user as a remote disk drive, a file server is a sophisticated device that not only stores files, but also manages them and maintains order as network users request files and make changes to these files.
final mode	The report processing mode of a processing mode of a program that updates or creates data records.
FTP server	A server that responds to requests for files via file transfer protocol.
header information	Information at the beginning of a table or form. Header information is used to identify or provide control information for the group of records that follows.
interface table	See Z table.
integration server	A server that facilitates interaction between diverse operating systems and applications across internal and external networked computer systems.
integrity test	A process used to supplement a company's internal balancing procedures by locating and reporting balancing problems and data inconsistencies.
interoperability model	A method for third-party systems to connect to or access JD Edwards EnterpriseOne.
in-your-face-error	In JD Edwards EnterpriseOne, a form-level property which, when enabled, causes the text of application errors to appear on the form.
IServer service	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.
jargon	An alternative data dictionary item description that JD Edwards EnterpriseOne appears based on the product code of the current object.
Java application server	A component-based server that resides in the middle-tier of a server-centric architecture. This server provides middleware services for security and state maintenance, along with data access and persistence.
JDBNET	A database driver that enables heterogeneous servers to access each other's data.
JDEBASE Database Middleware	A JD Edwards EnterpriseOne proprietary database middleware package that provides platform-independent APIs, along with client-to-server access.
JDECallObject	An API used by business functions to invoke other business functions.
jde.ini	A JD Edwards EnterpriseOne file (or member for iSeries) that provides the runtime settings required for JD Edwards EnterpriseOne initialization. Specific versions of the file or member must reside on every machine running JD Edwards EnterpriseOne. This includes workstations and servers.
JDEIPC	Communications programming tools used by server code to regulate access to the same data in multiprocess environments, communicate and coordinate between processes, and create new processes.

jde.log The main diagnostic log file of JD Edwards EnterpriseOne. This file is always located in the root directory on the primary drive and contains status and error messages from the startup and operation of JD Edwards EnterpriseOne.

JDENET A JD Edwards EnterpriseOne proprietary communications middleware package. This package 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 JD Edwards EnterpriseOne supported platforms.

Location Workbench An application that, during the Installation Workbench process, copies all locations that are defined in the installation plan from the Location Master table in the Planner data source to the system data source.

logic server A server in a distributed network that provides the business logic for an application program. In a typical configuration, pristine objects are replicated on to the logic server from the central server. The logic server, in conjunction with workstations, actually performs the processing required when JD Edwards EnterpriseOne software runs.

MailMerge Workbench An application that merges Microsoft Word 6.0 (or higher) word-processing documents with JD Edwards EnterpriseOne records to automatically print business documents. You can use MailMerge Workbench to print documents, such as form letters about verification of employment.

master business function (MBF) An interactive master file that serves as a central location for adding, changing, and updating information in a database. Master business functions pass information between data entry forms and the appropriate tables. These master functions provide a common set of functions that contain all of the necessary default and editing rules for related programs. MBFs contain logic that ensures the integrity of adding, updating, and deleting information from databases.

master table See published table.

matching document A document associated with an original document to complete or change a transaction. For example, in JD Edwards EnterpriseOne Financial Management, a receipt is the matching document of an invoice, and a payment is the matching document of a voucher.

media storage object Files that use one of the following naming conventions that are not organized into table format: Gxxx, xxxGT, or GTxxx.

message center A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user.

messaging adapter An interoperability model that enables third-party systems to connect to JD Edwards EnterpriseOne to exchange information through the use of messaging queues.

messaging server A server that handles messages that are sent for use by other programs using a messaging API. Messaging servers typically employ a middleware program to perform their functions.

named event rule (NER) Encapsulated, reusable business logic created using event rules, rather that C programming. NERs are also called business function event rules. NERs can be reused in multiple places by multiple programs. This modularity lends itself to streamlining, reusability of code, and less work.

nota fiscal In Brazil, a legal document that must accompany all commercial transactions for tax purposes and that must contain information required by tax regulations.

nota fiscal factura In Brazil, a nota fiscal with invoice information.

See also *nota fiscal.*

Object Configuration Manager (OCM)
In JD Edwards EnterpriseOne, the object request broker and control center for the runtime environment. OCM keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, OCM directs access to it using defaults and overrides for a given environment and user.

Object Librarian
A repository of all versions, applications, and business functions reusable in building applications. Object Librarian provides check-out and check-in capabilities for developers, and it controls the creation, modification, and use of JD Edwards EnterpriseOne objects. Object Librarian supports multiple environments (such as production and development) and enables objects to be easily moved from one environment to another.

Object Librarian merge
A process that blends any modifications to the Object Librarian in a previous release into the Object Librarian in a new release.

Open Data Access (ODA)
An interoperability model that enables you to use SQL statements to extract JD Edwards EnterpriseOne data for summarization and report generation.

Output Stream Access (OSA)
An interoperability model that enables you to set up an interface for JD Edwards EnterpriseOne to pass data to another software package, such as Microsoft Excel, for processing.

package
JD Edwards EnterpriseOne 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 installation program can find them. It is point-in-time snapshot of the central objects on the deployment server.

package build
A software application that facilitates the deployment of software changes and new applications to existing users. Additionally, in JD Edwards EnterpriseOne, a package build can be a compiled version of the software. When you upgrade your version of the ERP software, for example, you are said to take a package build.

Consider the following context: “Also, do not transfer business functions into the production path code until you are ready to deploy, because a global build of business functions done during a package build will automatically include the new functions.” The process of creating a package build is often referred to, as it is in this example, simply as “a package build.”

package location
The directory structure location for the package and its set of replicated objects. This is usually \\deployment server\release\path_code\package\package name. The subdirectories under this path are where the replicated objects for the package are placed. This is also referred to as where the package is built or stored.

Package Workbench
An application that, during the Installation Workbench process, transfers the package information tables from the Planner data source to the system-release number data source. It also updates the Package Plan detail record to reflect completion.

planning family
A means of grouping end items whose similarity of design and manufacture facilitates being planned in aggregate.

preference profile
The ability to define default values for specified fields for a user-defined hierarchy of items, item groups, customers, and customer groups.

print server
The interface between a printer and a network that enables network clients to connect to the printer and send their print jobs to it. A print server can be a computer, separate hardware device, or even hardware that resides inside of the printer itself.

pristine environment
A JD Edwards EnterpriseOne environment used to test unaltered objects with JD Edwards EnterpriseOne demonstration data or for training classes. You must have this environment so that you can compare pristine objects that you modify.

processing option	A data structure that enables users to supply parameters that regulate the running of a batch program or report. For example, you can use processing options to specify default values for certain fields, to determine how information appears or is printed, to specify date ranges, to supply runtime values that regulate program execution, and so on.
production environment	A JD Edwards EnterpriseOne environment in which users operate EnterpriseOne software.
production-grade file server	A file server that has been quality assurance tested and commercialized and that is usually provided in conjunction with user support services.
program temporary fix (PTF)	A representation of changes to JD Edwards EnterpriseOne software that your organization receives on magnetic tapes or disks.
project	In JD Edwards EnterpriseOne, a virtual container for objects being developed in Object Management Workbench.
promotion path	The designated path for advancing objects or projects in a workflow. The following is the normal promotion cycle (path): 11>21>26>28>38>01 In this path, *11* equals new project pending review, *21* equals programming, *26* equals QA test/review, *28* equals QA test/review complete, *38* equals in production, *01* equals complete. During the normal project promotion cycle, developers check objects out of and into the development path code and then promote them to the prototype path code. The objects are then moved to the productions path code before declaring them complete.
proxy server	A server that acts as a barrier between a workstation and the internet so that the enterprise can ensure security, administrative control, and caching service.
published table	Also called a master table, this is the central copy to be replicated to other machines. Residing on the publisher machine, the F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.
publisher	The server that is responsible for the published table. The F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.
pull replication	One of the JD Edwards EnterpriseOne methods for replicating data to individual workstations. Such machines are set up as pull subscribers using JD Edwards EnterpriseOne data replication tools. The only time that 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 F98DRPCN table.
QBE	An abbreviation for query by example. In JD Edwards EnterpriseOne, the QBE line is the top line on a detail area that is used for filtering data.
real-time event	A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and to provide notification to third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested notification when certain transactions occur.
refresh	A function used to modify JD Edwards EnterpriseOne software, or subset of it, such as a table or business data, so that it functions at a new release or cumulative update level, such as B73.2 or B73.2.1.
replication server	A server that is responsible for replicating central objects to client machines.
quote order	In JD Edwards Procurement and Subcontract Management, a request from a supplier for item and price information from which you can create a purchase order.

In JD Edwards Sales Order Management, item and price information for a customer who has not yet committed to a sales order.

selection

Found on JD Edwards EnterpriseOne menus, a selection represents functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.

Server Workbench

An application that, during the Installation Workbench process, copies the server configuration files from the Planner data source to the system-release number data source. It also updates the Server Plan detail record to reflect completion.

spot rate

An exchange rate entered at the transaction level. This rate overrides the exchange rate that is set up between two currencies.

Specification merge

A merge that comprises three merges: Object Librarian merge, Versions List merge, and Central Objects merge. The merges blend customer modifications with data that accompanies a new release.

specification

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

Specification Table Merge Workbench

An application that, during the Installation Workbench process, runs the batch applications that update the specification tables.

store-and-forward

The mode of processing that enables users who are disconnected from a server to enter transactions and then later connect to the server to upload those transactions.

subscriber table

Table F98DRSUB, which is stored on the publisher server with the F98DRPUB table and identifies all of the subscriber machines for each published table.

supplemental data

Any type of information that is not maintained in a master file. Supplemental data is usually additional information about employees, applicants, requisitions, and jobs (such as an employee's job skills, degrees, or foreign languages spoken). You can track virtually any type of information that your organization needs.

For example, in addition to the data in the standard master tables (the Address Book Master, Customer Master, and Supplier Master tables), you can maintain other kinds of data in separate, generic databases. These generic databases enable a standard approach to entering and maintaining supplemental data across JD Edwards EnterpriseOne systems.

table access management (TAM)

The JD Edwards EnterpriseOne component that handles the storage and retrieval of use-defined data. TAM stores information, such as data dictionary definitions; application and report specifications; event rules; table definitions; business function input parameters and library information; and data structure definitions for running applications, reports, and business functions.

Table Conversion Workbench

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

table conversion

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

table event rules

Logic that is attached to database triggers that runs whenever the action specified by the trigger occurs against the table. Although JD Edwards EnterpriseOne enables event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

terminal server

A server that enables terminals, microcomputers, and other devices to connect to a network or host computer or to devices attached to that particular computer.

three-tier processing	The task of entering, reviewing and approving, and posting batches of transactions in JD Edwards EnterpriseOne.
three-way voucher match	In JD Edwards Procurement and Subcontract Management, the process of comparing receipt information to supplier's invoices to create vouchers. In a three-way match, you use the receipt records to create vouchers.
transaction processing (TP) monitor	A monitor that controls data transfer between local and remote terminals and the applications that originated them. TP monitors also protect data integrity in the distributed environment and may include programs that validate data and format terminal screens.
transaction set	An electronic business transaction (electronic data interchange standard document) made up of segments.
trigger	One of several events specific to data dictionary items. You can attach logic to a data dictionary item that the system processes automatically when the event occurs.
triggering event	A specific workflow event that requires special action or has defined consequences or resulting actions.
two-way voucher match	In JD Edwards Procurement and Subcontract Management, the process of comparing purchase order detail lines to the suppliers' invoices to create vouchers. You do not record receipt information.
User Overrides merge	Adds new user override records into a customer's user override table.
variance	In JD Edwards Capital Asset Management, the difference between revenue generated by a piece of equipment and costs incurred by the equipment. In JD Edwards EnterpriseOne Project Costing and JD Edwards EnterpriseOne Manufacturing, the difference between two methods of costing the same item (for example, the difference between the frozen standard cost and the current cost is an engineering variance). Frozen standard costs come from the Cost Components table, and the current costs are calculated using the current bill of material, routing, and overhead rates.
Version List merge	The Versions List merge preserves any non-XJDE and non-ZJDE version specifications for objects that are valid in the new release, as well as their processing options data.
visual assist	Forms that can be invoked from a control via a trigger to assist the user in determining what data belongs in the control.
vocabulary override	An alternate description for a data dictionary item that appears on a specific JD Edwards EnterpriseOne form or report.
wchar_t	An internal type of a wide character. It is used for writing portable programs for international markets.
web application server	A web server that enables web applications to exchange data with the back-end systems and databases used in eBusiness transactions.
web server	A server that sends information as requested by a browser, using the TCP/IP set of protocols. A web server can do more than just coordination of requests from browsers; it can do anything a normal server can do, such as house applications or data. Any computer can be turned into a web server by installing server software and connecting the machine to the internet.
Windows terminal server	A multiuser server that enables terminals and minimally configured computers to display Windows applications even if they are not capable of running Windows software themselves. All client processing is performed centrally at the Windows

terminal server and only display, keystroke, and mouse commands are transmitted over the network to the client terminal device.

workbench A program that enables users to access a group of related programs from a single entry point. Typically, the programs that you access from a workbench are used to complete a large business process. For example, you use the JD Edwards EnterpriseOne Payroll Cycle Workbench (P07210) to access all of the programs that the system uses to process payroll, print payments, create payroll reports, create journal entries, and update payroll history. Examples of JD Edwards EnterpriseOne workbenches include Service Management Workbench (P90CD020), Line Scheduling Workbench (P3153), Planning Workbench (P13700), Auditor's Workbench (P09E115), and Payroll Cycle Workbench.

work day calendar In JD Edwards EnterpriseOne Manufacturing, a calendar that is used in planning functions that consecutively lists only working days so that component and work order scheduling can be done based on the actual number of work days available. A work day calendar is sometimes referred to as planning calendar, manufacturing calendar, or shop floor calendar.

workflow The automation of a business process, in whole or in 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 server that usually contains subsets of data replicated from a master network server. A workgroup server does not perform application or batch processing.

XAPI events A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and then calls third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested notification when the specified transactions occur to return a response.

XML CallObject An interoperability capability that enables you to call business functions.

XML Dispatch An interoperability capability that provides a single point of entry for all XML documents coming into JD Edwards EnterpriseOne for responses.

XML List An interoperability capability that enables you to request and receive JD Edwards EnterpriseOne database information in chunks.

XML Service An interoperability capability that enables you to request events from one JD Edwards EnterpriseOne system and receive a response from another JD Edwards EnterpriseOne system.

XML Transaction An interoperability capability that enables you to use a predefined transaction type to send information to or request information from JD Edwards EnterpriseOne. XML transaction uses interface table functionality.

XML Transaction Service (XTS) Transforms an XML document that is not in the JD Edwards EnterpriseOne format into an XML document that can be processed by JD Edwards EnterpriseOne. XTS then transforms the response back to the request originator XML format.

Z event A service that uses interface table functionality to capture JD Edwards EnterpriseOne transactions and provide notification to third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested to be notified when certain transactions occur.

Z table A working table where non-JD Edwards EnterpriseOne information can be stored and then processed into JD Edwards EnterpriseOne. Z tables also can be used to retrieve JD Edwards EnterpriseOne data. Z tables are also known as interface tables.

Z transaction Third-party data that is properly formatted in interface tables for updating to the JD Edwards EnterpriseOne database.

Index

A

additional documentation xii
additional features
  record locking 87
API, common library 3
API, database 5
API, JDEBASE 6
application fundamentals xi

B

Business Function Builder, DLLs 37
business functions
  calling APIs from 8
  creating C business functions 39
  creating event rule business functions 49
  debugging 89
  pessimistic record locking 88

C

caches
  calling JDECACHE APIs 23
  retrieving data from 24
  using cache business functions 22
  using programming standards 23
callback functions 13
Cdecl 9
comments, submitting xvi
common fields xvi
contact information xvi
cross-references xv
cursor, cache
  closing 33
  moving 31
  opening 29
  resetting 33
Customer Connection website xii

D

data structures
  JDEDATE 4
  MATH_NUMERIC 4
database locking 87
debugging business functions 89
DLLs 37
documentation
  printed xii
  related xii
  updates xii
DOM parser 10

H

handles 6

I

implementation guides
  ordering xii

J

JDB_InitUser API 26
jde.ini
  detecting record change 87
JDEB_InitBhvr API 23
JDEBase APIs, locking records 88
jdeCacheAdd API 23, 27, 29
jdeCacheCloseCursor API 29, 33
jdeCacheDelete API 32
jdeCacheDeleteAll API 32
jdeCacheFetch API 31
jdeCacheFetchPosition API 31, 32, 33
jdeCacheFetchPositionByRef API 33
JdeCacheGETNumCursors API 21
jdeCacheGetNumRecords API 21
jdeCacheInit API 21, 26, 27, 28
jdeCacheInitEx API 21
jdeCacheInitMultipleIndex 26
jdeCacheInitMultipleIndex API 21, 26
jdeCACHEINITMultipleIndex API 23
jdeCacheInitMultipleIndexEx API 21
jdeCACHEINITMultipleIndexEx API 23
jdeCacheInitMultipleIndexUser API 22
jdeCacheInitUser API 22
jdeCacheOpenCursor API 23, 29
jdeCacheResetCursor API 33
jdeCacheTerminate API 28
jdeCacheTerminateALL API 28
jdeCacheUpdate API 32
jdeCachInit API 23
jdeCachInitEx API 23
JDEDATE 4

JDEKRNL 20

M

MATH_NUMERIC 4

N

native locking strategy 87
notes xv
null pointer errors 90

O

ODBC 5
optimistic locking 87
output errors 90

P

parsers
  DOM 10
  SAX 10
PeopleCode, typographical conventions xiv
pessimistic locking
  business functions 88
  overview 88
  transaction boundary 88
prerequisites xi
printed documentation xii

R

record locking
  native locking strategy 87
  optimistic locking 87
  pessimistic locking 88
  PSFTBase APIs 88
related documentation xii

S

SAX parser 10
Stdcall 8
suggestions, submitting xvi

T

transaction boundary, pessimistic locking 88
typographical conventions xiv

U

unhandled exception 89

V

Visual Basic program 9
visual cues xv

W

warnings xv
workstation jde.ini, record change detection 87

X

XercesWrapper 10