PeopleSoft®

EnterpriseOne Tools 8.94 PeopleBook: Development Tools: APIs and Business Functions

November 2004

EnterpriseOne Tools 8.94 PeopleBook: Development Tools: APIs and Business Functions
SKU E1_TOOLS8.94TBF-B 1104

Open Source Disclosure

PeopleSoft takes no responsibility for its use or distribution of any open source or shareware software or documentation and disclaims any and all liability or damages resulting from use of said software or documentation. The following open source software may be used in PeopleSoft products and the following disclaimers are provided.

Apache Software Foundation

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenSSL

Copyright (c) 1998-2003 The OpenSSL Project. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SSLeay

Copyright (c) 1995-1998 Eric Young. All rights reserved.

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Loki Library

Copyright (c) 2001 by Andrei Alexandrescu. This code accompanies the book:

Alexandrescu, Andrei. "Modern C++ Design: Generic Programming and Design Patterns Applied". Copyright (c) 2001. Addison-Wesley. Permission to use, copy, modify, distribute and sell this software for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

Contents

About This PeopleBook

PeopleBooks provide you with the information that you need to implement and use PeopleSoft applications.

This preface discusses:

- PeopleSoft application prerequisites.
- PeopleSoft application fundamentals.
- Documentation updates and printed documentation.
- Additional resources.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common elements in PeopleBooks.

Note. PeopleBooks document only page elements, such as fields and check boxes, that require additional explanation. If a page 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 elements for the section, chapter, PeopleBook, or product line. Elements that are common to all PeopleSoft applications are defined in this preface.

PeopleSoft Application Prerequisites

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

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

You should be familiar with navigating the system and adding, updating, and deleting information by using PeopleSoft menus, and pages, 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 PeopleSoft applications most effectively.

PeopleSoft Application Fundamentals

Each application PeopleBook provides implementation and processing information for your PeopleSoft 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 PeopleBook. Most PeopleSoft product lines have a version of the application fundamentals PeopleBook. The preface of each PeopleBook identifies the application fundamentals PeopleBooks that are associated with that PeopleBook.

The application fundamentals PeopleBook consists of important topics that apply to many or all PeopleSoft applications across one or more product lines. 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 PeopleBooks. 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 the PeopleSoft Customer Connection website. Through the Documentation section of PeopleSoft Customer Connection, you can download files to add to your PeopleBook Library. You'll find a variety of useful and timely materials, including updates to the full PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM.

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

See Also

PeopleSoft Customer Connection, https://www.peoplesoft.com/corp/en/login.jsp

Ordering Printed Documentation

You can order printed, bound volumes of the complete PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM. PeopleSoft makes printed documentation available for each major release shortly after the software is shipped. Customers and partners can order printed PeopleSoft documentation by using any of these methods:

- Web
- Telephone
- Email

Web

From the Documentation section of the PeopleSoft Customer Connection website, access the PeopleBooks Press website under the Ordering PeopleBooks topic. The PeopleBooks Press website is a joint venture between PeopleSoft and MMA Partners, the book print vendor. Use a credit card, money order, cashier's check, or purchase order to place your order.

Telephone

Contact MMA Partners at 877 588 2525.

Email

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

See Also

PeopleSoft Customer Connection, https://www.peoplesoft.com/corp/en/login.jsp

Additional Resources

The following resources are located on the PeopleSoft Customer Connection website:

Resource	Navigation
Application maintenance information	Updates + Fixes
Business process diagrams	Support, Documentation, Business Process Maps
Interactive Services Repository	Interactive Services Repository
Hardware and software requirements	Implement, Optimize + Upgrade, Implementation Guide, Implementation Documentation & Software, Hardware and Software Requirements
Installation guides	Implement, Optimize + Upgrade, Implementation Guide, Implementation Documentation & Software, Installation Guides and Notes
Integration information	Implement, Optimize + Upgrade, Implementation Guide, Implementation Documentation and Software, Pre-built Integrations for PeopleSoft Enterprise and PeopleSoft EnterpriseOne Applications
Minimum technical requirements (MTRs) (EnterpriseOne only)	Implement, Optimize + Upgrade, Implementation Guide, Supported Platforms
PeopleBook documentation updates	Support, Documentation, Documentation Updates
PeopleSoft support policy	Support, Support Policy
Prerelease notes	Support, Documentation, Documentation Updates, Category, Prerelease 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 PeopleBooks:

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 PeopleSoft 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

PeopleBooks contain the following visual cues.

Notes

Notes indicate information that you should pay particular attention to as you work with the PeopleSoft 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

PeopleBooks 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 PeopleBooks:

- 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 PeopleBooks:

- 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 PeopleBooks and other PeopleSoft reference and training materials. Please send your suggestions to:

PeopleSoft Product Documentation Manager PeopleSoft, Inc. 4460 Hacienda Drive Pleasanton, CA 94588

Or send email comments to doc@peoplesoft.com.

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

Common Elements Used in PeopleBooks

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 allows 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 it 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. PeopleSoft 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. PeopleSoft 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.

APIs and Business Functions Preface

This preface discusses the APIs and Business Functions PeopleBook.

PeopleSoft Products

This PeopleBook refers to this PeopleSoft product line: PeopleSoft EnterpriseOne Tools.

PeopleSoft Tools API and Business Functions

This PeopleBook covers APIs and Business Functions, prepackaged code objects use by the PeopleSoft EnterpriseOne Tools suite. Its chapters describe APIs and business functions in general, how to call them, and how to create business functions using Business Function Builder.

CHAPTER 1

Getting Started with PeopleSoft Tools APIs and Business Functions

This chapter provides an overview of APIs and business functions.

PeopleSoft Tools APIs and Business Functions Overview

Use business functions to create complex, reusable routines in C. Business functions can call APIs directly, and can in turn be invoked from event rules (ER).

Other Sources of Information

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

See Also

About These PeopleBooks Preface

PeopleSoft EnterpriseOne Tools Business Functions and APIs Implementation

To use business functions with the PeopleSoft EnterpriseOne applications, these tasks must be completed first:

- You must have a valid PeopleSoft EnterpriseOne account.

  Depending on how security has been configured, you might need one or more roles assigned to you so that you can access Object Management Workbench (OMW), the PeopleSoft EnterpriseOne databases, and so forth.
- OMW must be configured with transfer activity rules and allowed actions so that application development can occur.
- At a minimum, you must have a default project in OMW to which you have been added in the role of Developer.

CHAPTER 2

Working with APIs

This chapter provides an overview of the EnterpriseOne implementation of APIs and discusses how to call APIs.

Understanding APIs

This section discusses APIs.

APIs

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

PeopleSoft 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 PeopleSoft EnterpriseOne to manipulate the variables of these data types.

MATH_NUMERIC Data Type

The MATH_NUMERIC data type exclusively represents all numeric values in PeopleSoft 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 PeopleSoft 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

PeopleSoft 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).

PeopleSoft EnterpriseOne ODBC

PeopleSoft 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 PeopleSoft 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 PeopleSoft EnterpriseOne database function call. The driver manager processes the request or passes it to an appropriate driver.

PeopleSoft 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 PeopleSoft 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 PeopleSoft 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 your particular API.
typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR, LPTSTR);
// Now create a variable for your 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⇒
 your 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 your particular API.
typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR, LPTSTR);
// Now create a variable for your 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 PeopleSoft EnterpriseOne Software

You can call a Visual Basic program from a PeopleSoft EnterpriseOne business function and pass a parameter from the Visual Basic program to the PeopleSoft 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 PeopleSoft 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.

See Also

EnterpriseOne Tools 8.94 PeopleBook: Configurable Network Computing Implementation, "Object Configuration Manager"

Understanding the SAX Parser

This chapter discusses the SAX parser and provides examples for its use.

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 re-reading 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.

Understanding Caching

This chapter discusses caching and the JDECACHE API.

Caching

Caching is a process that stores a local copy of frequently accessed content of remote objects. Caching can improve performance. 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 PeopleSoft 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.

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**
- **jdeCacheInitMultipleIndex**
- **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 **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**

Working with JDECACHE

This chapter provides an overview of JDECACHE standards 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.

Prerequisites

Before you can use JDECACHE, you must initialize a cache. You must define an index before you initialize a cache. The index specifies to the cache which fields in a record are used to uniquely identify a cache record. You must create a separate cache for each group of data that an index references.

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.

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** or **jdeCacheInitMultipleIndex**.
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 Indices

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** or **jdeCacheInitMultipleIndex** 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** and **jdeCacheInitMultipleIndex** 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** 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 nInt1
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**, and JDECACHE creates this Index Definition Structure for internal use. The 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** for that cache in number, order, offset positions, and type. Therefore, if a field that was used in the index passed to **jdeCacheInit** 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** whenever you call an API that requires an index structure.

The next example illustrates why the index offsets must be specified for the **jdeCacheInit** 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**.

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**. 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** 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** will be retained. Always call **jdeCacheInit** with the same index each time that you call it with the same cache name. If you call **jdeCacheInit** 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** or **jdeCacheInitMultipleIndex**, 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** or **jdeCacheInitMultipleIndex** calls must be terminated with a **jdeCacheTerminateAll**.

A **jdeCacheTerminate** call terminates the most recent corresponding **jdeCacheInit**. This means that the same cache can be used in nested business functions. In each function, perform a **jdeCacheInit** 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** calls. In contrast, one call to **jdeCacheTerminateAll** destroys the cache from memory regardless of the number of **jdeCacheInit** or **jdeCacheInitMultipleIndex** 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** 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** again with the same index and cache name as in the initial call to **jdeCacheInit**. 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** 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 **jdeCacheFetchPostion** 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** and populated by a call to **jdeCacheAdd** before a cursor can be opened.

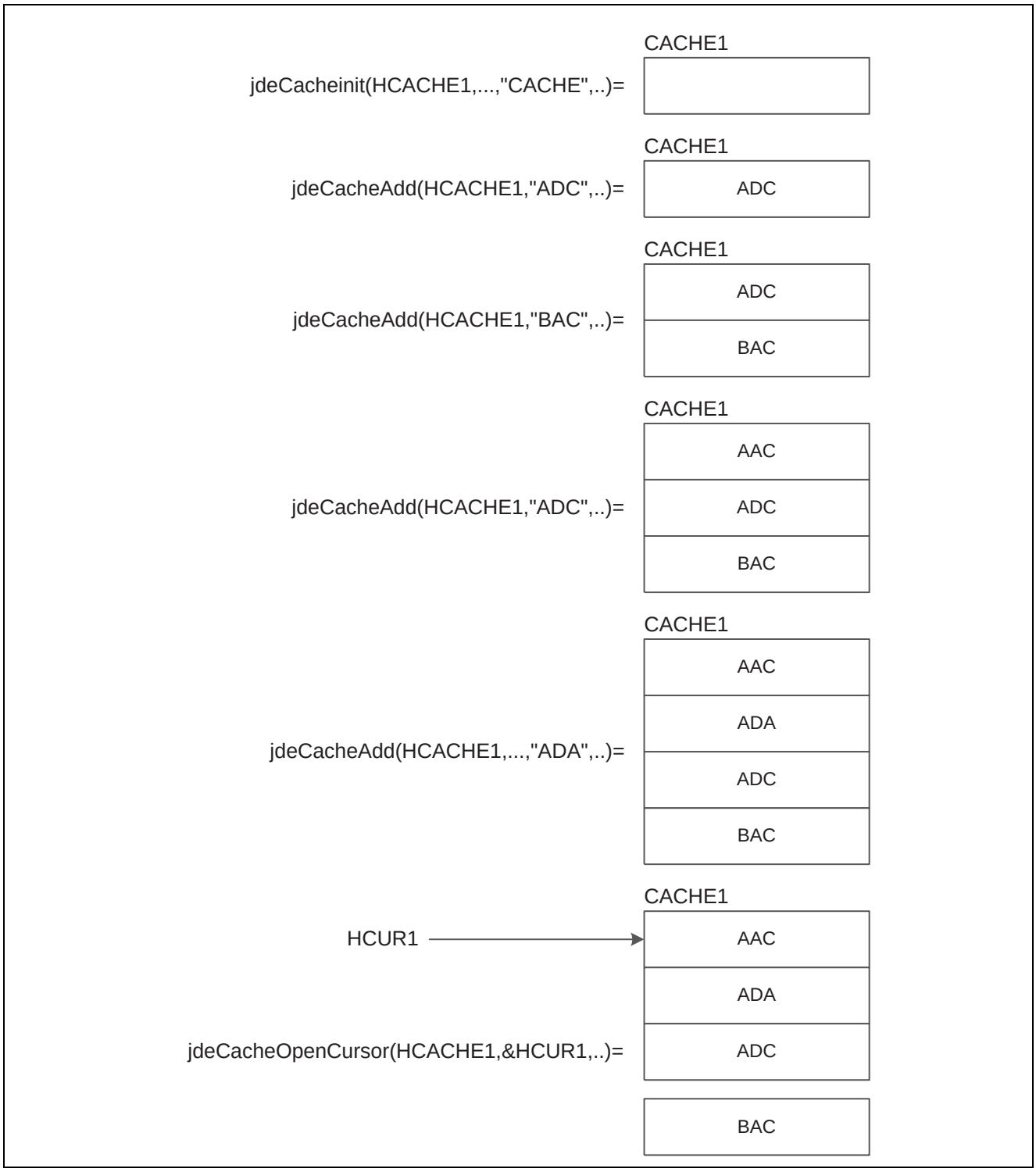
You can obtain multiple cursors to a cache by calling **jdeCacheOpenCursor** and passing different HJDECURSOR handles. In a multiple cursor environment, all the cursors are independent of each other.

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 enables up to 100 open cursors 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 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**.)

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** 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**, 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

Business Functions

This chapter discusses both C business functions and named event rules, and includes information about master business functions, Business Function Builder, and business function documentation.

Understanding Business Functions

You can use business functions to enhance PeopleSoft 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 the needs.

- C programming code.

  PeopleSoft 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 PeopleSoft 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.

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 PeopleSoft 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

DLL Name	Functional Group
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
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

DLL Name	Functional Group
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.

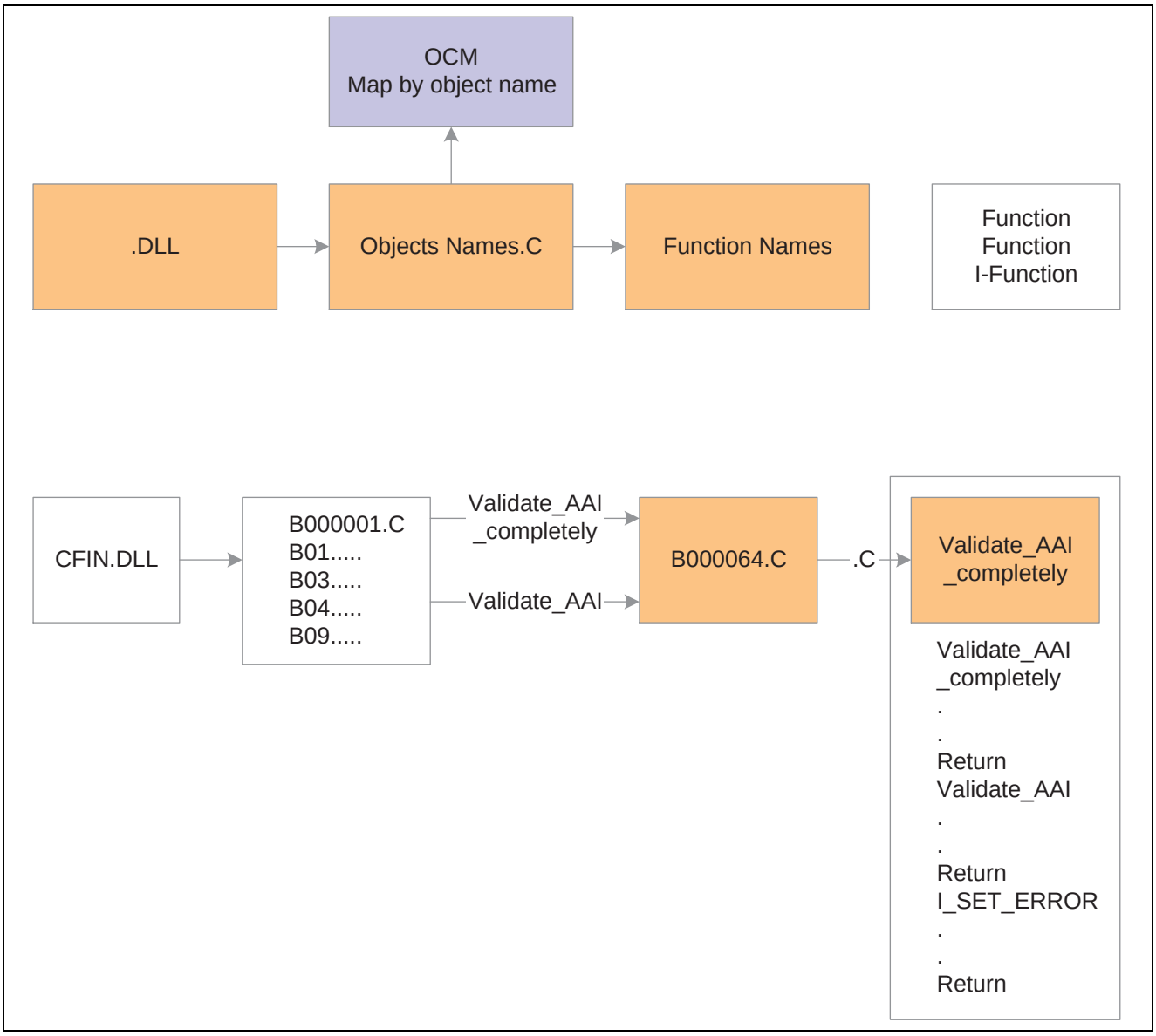
  Note. Any business function, whether it uses C or NERs as its source language, must have a defined data structure to send or receive parameters to or from applications. You can create a DSTR data structure object, or choose an existing object type to work with in OMW. You can also create data structures 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

See Also

EnterpriseOne Tools 8.94 PeopleBook: Configurable Network Computing Implementation, "Object Configuration Manager"

C Business Functions

PeopleSoft 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 PeopleSoft, Inc., 2003
*
* This unpublished material is proprietary to PeopleSoft, Inc.
* 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 PeopleSoft, Inc.
**************************************************************************/
#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 PeopleSoft 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 PeopleSoft 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 PeopleSoft 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 Numberic
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
```

Working with 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-PeopleSoft 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				

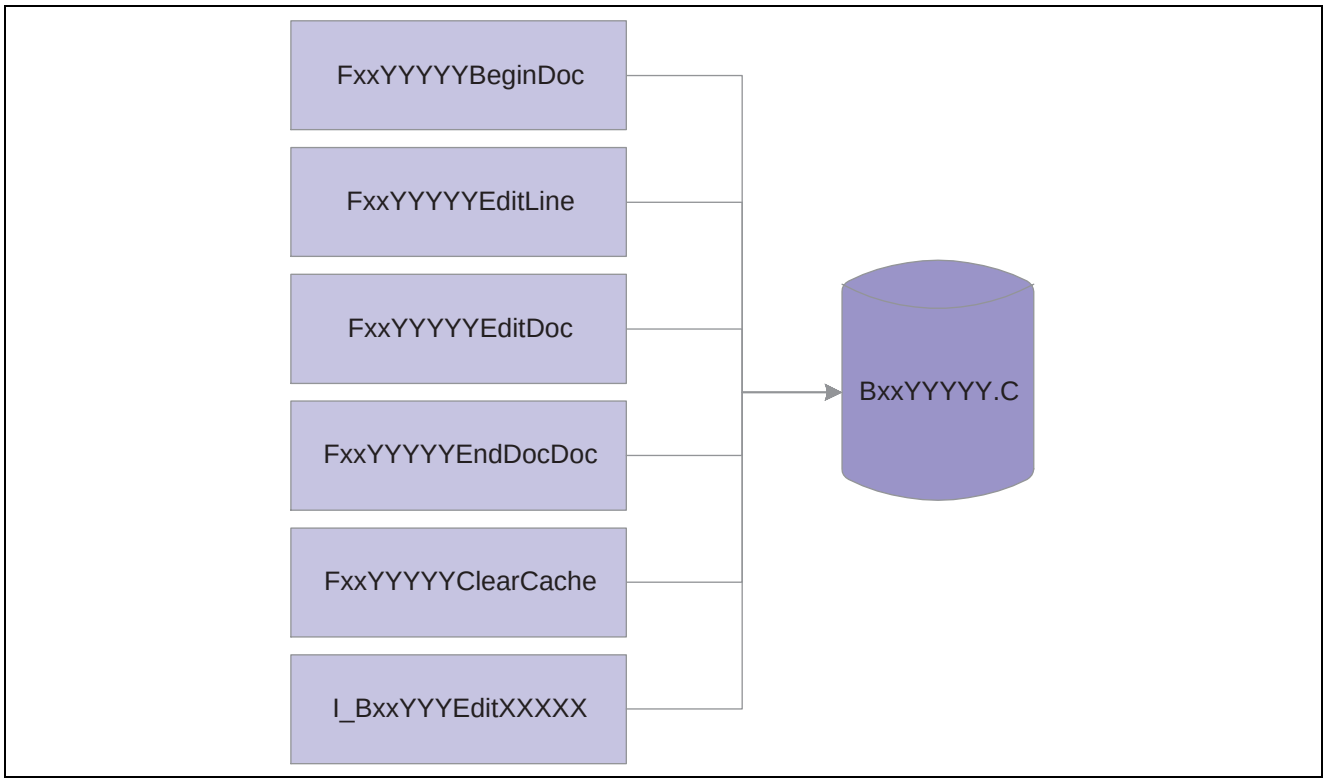
This table explains the fields:

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*xxxxxxxxx[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.

Creating Transaction Master Business Functions

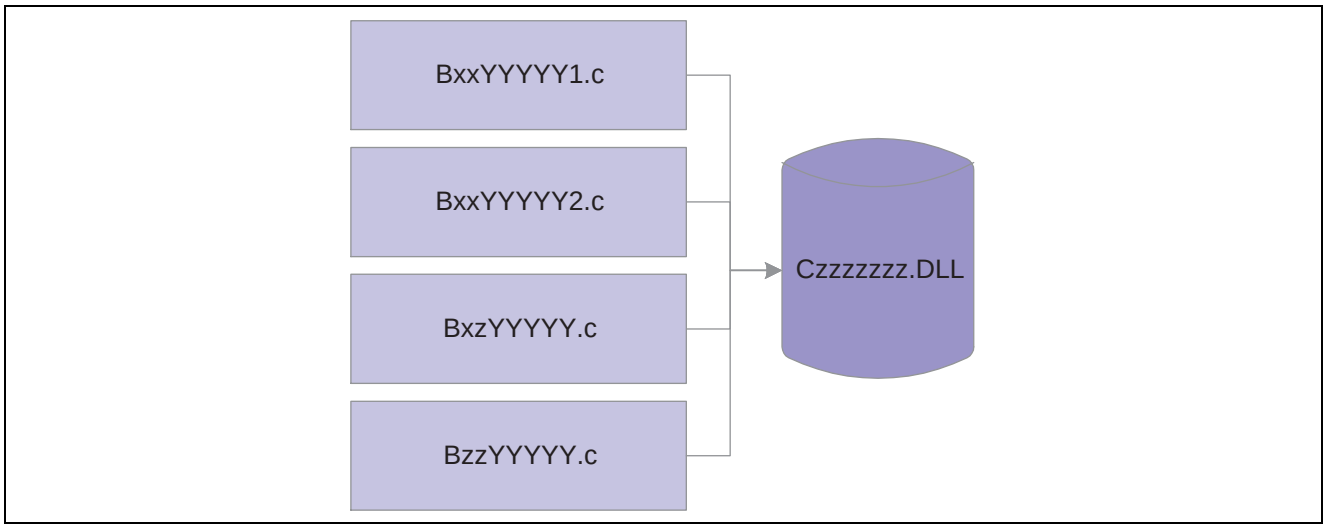
These flowcharts illustrate how transaction master business functions are built.

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.
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.

Name	Alias	I/O	Description
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.
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
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	

Name	Alias	I/O	Description
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.

Name	Alias	I/O	Description
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.

Building Transaction Master Business Functions

This chapter 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.

Understanding 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 PeopleSoft 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 PeopleSoft 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 Cod.e 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
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 choose 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 PeopleSoft EnterpriseOne business function programming standards. Creating a business function involves these overall tasks:

- Use 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 PeopleSoft 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 PeopleSoft EnterpriseOne CCUSTOM.DLL, where CUSTOM 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 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 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 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 Buildlog contains the same information as the Build Output form in 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.

Option	Result
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 PeopleSoft EnterpriseOne consolidated DLL.

Using the Utility Programs

The Tool 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 PeopleSoft 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.

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. 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, 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, 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, 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, Business 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, Business Function Builder notes that the DLL was built with errors. If a fatal error occurs, 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*****. 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.

Resolving Errors

You can use 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 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 Business Function Builder. Use 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 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 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.

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.

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 PeopleSoft 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 choose 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 PeopleSoft EnterpriseOne foundation.

   - *Foundation*

     The foundation table lists all registered PeopleSoft EnterpriseOne foundations. Choose a foundation from this table.

   - *Pick Directory*

     The PeopleSoft EnterpriseOne foundation exists in a directory on the file server. You specify the directory. PeopleSoft 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 choose *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, Business Function Builder does not build a business function if it encounters any warnings.

   - Clear Output Destination Before Build

     If you select this option, 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, Business Function Builder builds all DLLs. If you select this option, you can click the Select button and choose 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, 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, 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.

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 functions 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 choose 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 PeopleSoft EnterpriseOne software directory.

APPENDIX A

PeopleSoft EnterpriseOne APIs

This appendix describes the public APIs available for PeopleSoft EnterpriseOne.

General APIs

This section discusses APIs that can used in a number of different PeopleSoft 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, "PeopleSoft EnterpriseOne APIs," jdeCreateGuidString, page 80
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGuidCompare, page 80
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGuidToString, page 81

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, “PeopleSoft EnterpriseOne APIs,” jdeCreateGuid, page 79
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGuidCompare, page 80
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGuidToString, page 81

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, “PeopleSoft EnterpriseOne APIs,” jdeCreateGuid, page 79
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeCreateGuidString, page 80
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGuidToString, page 81

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, "PeopleSoft EnterpriseOne APIs," jdeCreateGuid, page 79
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeCreateGuidString, page 80
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGuidCompare, page 80

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 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 the API succeeded.
0	JDEENCRYPT_FAILURE. Indicates 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 hRequest0;
```

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);
```

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 PeopleSoft 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 PeopleSoft 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 indices synchronized with changes to the corresponding PeopleSoft 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 incremental text search index build completed successfully.
TEXTSEARCH_INDEXING_FAIL	The incremental 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 incremental text search clear completed successfully.

Value	Description
TEXTSEARCH_INDEXING_FAIL	The incremental text search clear 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 text search index optimization completed successfully.
TEXTSEARCH_INDEXING_FAIL	The text search index optimization failed.

Example

```
TEXTSEARCH_INDEXING_RESULT nResult;

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

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 99
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 103
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 108
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_OpenTable, page 112
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 114
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGT_CloseTable, page 98
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 103
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 108
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_OpenTable, page 112
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 114
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGT_CloseTable, page 98
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 99
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 108
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_OpenTable, page 112
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 114
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118

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 the API succeeded.
JDEDB_FAILED	Indicates 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, "PeopleSoft EnterpriseOne APIs," jdeGT_CloseTable, page 98
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 99
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGT_FetchData/jdeGT_FetchDataEx, page 103
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGT_OpenTable, page 112
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 114
Appendix A, "PeopleSoft EnterpriseOne APIs," jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGT_OpenTable, page 112
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 99
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 103
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 108
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 114
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118

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.

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGT_CloseTable, page 98
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 99
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 103
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 108
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_OpenTable, page 112
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118

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.

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGT_UpdateData/jdeGT_UpdateDataKeyStr, page 118
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_DeleteData/jdeGT_DeleteDataKeyStr, page 99
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_FetchData/jdeGT_FetchDataEx, page 103
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_InsertData/jdeGT_InsertDataKeyStr, page 108
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_OpenTable, page 112
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGT_SelectData/jdeGT_SelectDataKeyStr, page 114

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.
ITotalRec	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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
ITotalRec	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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
ITotalRec	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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
ITotalRec	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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
ITotalRec	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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
ITotalRec	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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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]	

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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.

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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.

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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.

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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]	
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.
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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeValidateGTExist/jdeValidateGTExistWithKeyStr, page 223

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.

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 the API succeeded.
JDEDB_FAILED	Indicates 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, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_AllMOType/ jdeGTAddUpdate_AllMOTypeKeyStr/ jdeGTAddUpdate_AllMOTypeWithLang, page 122
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_HTML/jdeGTAddUpdate_HTMLKeyStr, page 126
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Image/jdeGTAddUpdate_ImageKeyStr, page 129
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_OLE/jdeGTAddUpdate_OLEKeyStr, page 133
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Shortcut/jdeGTAddUpdate_ShortcutKeyStr, page 136
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Text/jdeGTAddUpdate_TextKeyStr, page 140
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTAddUpdate_Vendor/jdeGTAddUpdate_VendorKeyStr, page 143
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllHTML/jdeGTDelete_AllHTMLKeyStr, page 147
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllImage/jdeGTDelete_AllImageKeyStr, page 149
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllMOType/jdeGTDelete_AllMOTypeStr, page 152
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllOLE/jdeGTDelete_AllOLEKeyStr, page 155
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllShortcut/jdeGTDelete_AllShortcutKeyStr, page 158
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllText/jdeGTDelete_AllTextKeyStr, page 161
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTDelete_AllVendor/jdeGTDelete_AllVendorKeyStr, page 164
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTFreeMOData, page 188
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_AllMOType/jdeGTGet_AllMOTypeKeyStr, page 191
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_GenericText/jdeGTGet_GenericTextKeyStr, page 195
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_HTML/jdeGTGet_HTMLKeyStr, page 198
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Image/jdeGTGet_ImageKeyStr, page 202
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_OLE/jdeGTGet_OLEKeyStr, page 205
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_RTFText/jdeGTGet_RTFTextKeyStr, page 209
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Shortcut/jdeGTGet_ShortcutKeyStr, page 212
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGet_Vendor/jdeGTGet_VendorKeyStr, page 216
Appendix A, “PeopleSoft EnterpriseOne APIs,” jdeGTGetCount/jdeGTGetCountKeyStr, page 220

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@peoplesoft.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 thread-safe, 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 PeopleSoft Terms

absence entitlement This element defines rules for granting paid time off for valid absences, such as sick time, vacation, and maternity leave. An absence entitlement element defines the entitlement amount, frequency, and entitlement period.

absence take This element defines the conditions that must be met before a payee is entitled to take paid time off.

academic career In PeopleSoft Enterprise Campus Solutions, all course work that a student undertakes at an academic institution and that is grouped in a single student record. For example, a university that has an undergraduate school, a graduate school, and various professional schools might define several academic careers—an undergraduate career, a graduate career, and separate careers for each professional school (law school, medical school, dental school, and so on).

academic institution In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.

academic organization In PeopleSoft Enterprise Campus Solutions, an entity that is part of the administrative structure within an academic institution. At the lowest level, an academic organization might be an academic department. At the highest level, an academic organization can represent a division.

academic plan In PeopleSoft Enterprise Campus Solutions, an area of study—such as a major, minor, or specialization—that exists within an academic program or academic career.

academic program In PeopleSoft Enterprise Campus Solutions, the entity to which a student applies and is admitted and from which the student graduates.

accounting class In PeopleSoft Enterprise Performance Management, the accounting class defines how a resource is treated for generally accepted accounting practices. The Inventory class indicates whether a resource becomes part of a balance sheet account, such as inventory or fixed assets, while the Non-inventory class indicates that the resource is treated as an expense of the period during which it occurs.

accounting date The accounting date indicates when a transaction is recognized, as opposed to the date the transaction actually occurred. The accounting date and transaction date can be the same. The accounting date determines the period in the general ledger to which the transaction is to be posted. You can only select an accounting date that falls within an open period in the ledger to which you are posting. The accounting date for an item is normally the invoice date.

accounting split The accounting split method indicates how expenses are allocated or divided among one or more sets of accounting ChartFields.

accumulator You use an accumulator to store cumulative values of defined items as they are processed. You can accumulate a single value over time or multiple values over time. For example, an accumulator could consist of all voluntary deductions, or all company deductions, enabling you to accumulate amounts. It allows total flexibility for time periods and values accumulated.

action reason The reason an employee's job or employment information is updated. The action reason is entered in two parts: a personnel action, such as a promotion, termination, or change from one pay group to another—and a reason for that action. Action reasons are used by PeopleSoft Human Resources, PeopleSoft Benefits Administration,

PeopleSoft Stock Administration, and the COBRA Administration feature of the Base Benefits business process.

action template In PeopleSoft Receivables, outlines a set of escalating actions that the system or user performs based on the period of time that a customer or item has been in an action plan for a specific condition.

activity In PeopleSoft Enterprise Learning Management, an instance of a catalog item (sometimes called a class) that is available for enrollment. The activity defines such things as the costs that are associated with the offering, enrollment limits and deadlines, and waitlisting capacities.

In PeopleSoft Enterprise Performance Management, the work of an organization and the aggregation of actions that are used for activity-based costing.

In PeopleSoft Project Costing, the unit of work that provides a further breakdown of projects—usually into specific tasks.

In PeopleSoft Workflow, a specific transaction that you might need to perform in a business process. Because it consists of the steps that are used to perform a transaction, it is also known as a step map.

address usage In PeopleSoft Enterprise Campus Solutions, a grouping of address types defining the order in which the address types are used. For example, you might define an address usage code to process addresses in the following order: billing address, dormitory address, home address, and then work address.

adjustment calendar In PeopleSoft Enterprise Campus Solutions, the adjustment calendar controls how a particular charge is adjusted on a student's account when the student drops classes or withdraws from a term. The charge adjustment is based on how much time has elapsed from a predetermined date, and it is determined as a percentage of the original charge amount.

administrative function In PeopleSoft Enterprise Campus Solutions, a particular functional area that processes checklists, communication, and comments. The administrative function identifies which variable data is added to a person's checklist or communication record when a specific checklist code, communication category, or comment is assigned to the student. This key data enables you to trace that checklist, communication, or comment back to a specific processing event in a functional area.

admit type In PeopleSoft Enterprise Campus Solutions, a designation used to distinguish first-year applications from transfer applications.

agreement In PeopleSoft eSettlements, provides a way to group and specify processing options, such as payment terms, pay from a bank, and notifications by a buyer and supplier location combination.

allocation rule In PeopleSoft Enterprise Incentive Management, an expression within compensation plans that enables the system to assign transactions to nodes and participants. During transaction allocation, the allocation engine traverses the compensation structure from the current node to the root node, checking each node for plans that contain allocation rules.

alternate account A feature in PeopleSoft General Ledger that enables you to create a statutory chart of accounts and enter statutory account transactions at the detail transaction level, as required for recording and reporting by some national governments.

analysis database In PeopleSoft Enterprise Campus Solutions, database tables that store large amounts of student information that may not appear in standard report formats. The analysis database tables contain keys for all objects in a report that an application program can use to reference other student-record objects that are not contained in the printed report. For instance, the analysis database contains data on courses that are considered for satisfying a requirement but that are rejected. It also contains information on

	courses captured by global limits. An analysis database is used in PeopleSoft Enterprise Academic Advisement.
AR specialist	Abbreviation for *receivables specialist.* In PeopleSoft Receivables, an individual in who tracks and resolves deductions and disputed items.
arbitration plan	In PeopleSoft Enterprise Pricer, defines how price rules are to be applied to the base price when the transaction is priced.
assessment rule	In PeopleSoft Receivables, a user-defined rule that the system uses to evaluate the condition of a customer's account or of individual items to determine whether to generate a follow-up action.
asset class	An asset group used for reporting purposes. It can be used in conjunction with the asset category to refine asset classification.
attribute/value pair	In PeopleSoft Directory Interface, relates the data that makes up an entry in the directory information tree.
audience	In PeopleSoft Enterprise Campus Solutions, a segment of the database that relates to an initiative, or a membership organization that is based on constituent attributes rather than a dues-paying structure. Examples of audiences include the Class of '65 and Undergraduate Arts & Sciences.
authentication server	A server that is set up to verify users of the system.
base time period	In PeopleSoft Business Planning, the lowest level time period in a calendar.
benchmark job	In PeopleSoft Workforce Analytics, a benchmark job is a job code for which there is corresponding salary survey data from published, third-party sources.
billing career	In PeopleSoft Enterprise Campus Solutions, the one career under which other careers are grouped for billing purposes if a student is active simultaneously in multiple careers.
bio bit or bio brief	In PeopleSoft Enterprise Campus Solutions, a report that summarizes information stored in the system about a particular constituent. You can generate standard or specialized reports.
book	In PeopleSoft Asset Management, used for storing financial and tax information, such as costs, depreciation attributes, and retirement information on assets.
branch	A tree node that rolls up to nodes above it in the hierarchy, as defined in PeopleSoft Tree Manager.
budgetary account only	An account used by the system only and not by users; this type of account does not accept transactions. You can only budget with this account. Formerly called "system-maintained account."
budget check	In commitment control, the processing of source transactions against control budget ledgers, to see if they pass, fail, or pass with a warning.
budget control	In commitment control, budget control ensures that commitments and expenditures don't exceed budgets. It enables you to track transactions against corresponding budgets and terminate a document's cycle if the defined budget conditions are not met. For example, you can prevent a purchase order from being dispatched to a vendor if there are insufficient funds in the related budget to support it.
budget period	The interval of time (such as 12 months or 4 quarters) into which a period is divided for budgetary and reporting purposes. The ChartField allows maximum flexibility to define operational accounting time periods without restriction to only one calendar.

business event In PeopleSoft Receivables, defines the processing characteristics for the Receivable Update process for a draft activity.

In PeopleSoft Sales Incentive Management, an original business transaction or activity that may justify the creation of a PeopleSoft Enterprise Incentive Management event (a sale, for example).

business unit A corporation or a subset of a corporation that is independent with regard to one or more operational or accounting functions.

buyer In PeopleSoft eSettlements, an organization (or business unit, as opposed to an individual) that transacts with suppliers (vendors) within the system. A buyer creates payments for purchases that are made in the system.

campus In PeopleSoft Enterprise Campus Solutions, an entity that is usually associated with a distinct physical administrative unit, that belongs to a single academic institution, that uses a unique course catalog, and that produces a common transcript for students within the same academic career.

catalog item In PeopleSoft Enterprise Learning Management, a specific topic that a learner can study and have tracked. For example, "Introduction to Microsoft Word." A catalog item contains general information about the topic and includes a course code, description, categorization, keywords, and delivery methods. A catalog item can have one or more learning activities.

catalog map In PeopleSoft Catalog Management, translates values from the catalog source data to the format of the company's catalog.

catalog partner In PeopleSoft Catalog Management, shares responsibility with the enterprise catalog manager for maintaining catalog content.

categorization Associates partner offerings with catalog offerings and groups them into enterprise catalog categories.

category In PeopleSoft Enterprise Campus Solutions, a broad grouping to which specific comments or communications (contexts) are assigned. Category codes are also linked to 3C access groups so that you can assign data-entry or view-only privileges across functions.

channel In PeopleSoft MultiChannel Framework, email, chat, voice (computer telephone integration [CTI]), or a generic event.

ChartField A field that stores a chart of accounts, resources, and so on, depending on the PeopleSoft application. ChartField values represent individual account numbers, department codes, and so forth.

ChartField balancing You can require specific ChartFields to match up (balance) on the debit and the credit side of a transaction.

ChartField combination edit The process of editing journal lines for valid ChartField combinations based on user-defined rules.

ChartKey One or more fields that uniquely identify each row in a table. Some tables contain only one field as the key, while others require a combination.

checkbook In PeopleSoft Promotions Management, enables you to view financial data (such as planned, incurred, and actual amounts) that is related to funds and trade promotions.

checklist code In PeopleSoft Enterprise Campus Solutions, a code that represents a list of planned or completed action items that can be assigned to a staff member, volunteer, or unit. Checklists enable you to view all action assignments on one page.

class In PeopleSoft Enterprise Campus Solutions, a specific offering of a course component within an academic term.

See also *course.*

Class ChartField A ChartField value that identifies a unique appropriation budget key when you combine it with a fund, department ID, and program code, as well as a budget period. Formerly called *sub-classification.*

clearance In PeopleSoft Enterprise Campus Solutions, the period of time during which a constituent in PeopleSoft Contributor Relations is approved for involvement in an initiative or an action. Clearances are used to prevent development officers from making multiple requests to a constituent during the same time period.

clone In PeopleCode, to make a unique copy. In contrast, to *copy* may mean making a new reference to an object, so if the underlying object is changed, both the copy and the original change.

cohort In PeopleSoft Enterprise Campus Solutions, the highest level of the three-level classification structure that you define for enrollment management. You can define a cohort level, link it to other levels, and set enrollment target numbers for it.

See also *population*and *division*.

collection To make a set of documents available for searching in Verity, you must first create at least one collection. A collection is set of directories and files that allow search application users to use the Verity search engine to quickly find and display source documents that match search criteria. A collection is a set of statistics and pointers to the source documents, stored in a proprietary format on a file server. Because a collection can only store information for a single location, PeopleSoft maintains a set of collections (one per language code) for each search index object.

collection rule In PeopleSoft Receivables, a user-defined rule that defines actions to take for a customer based on both the amount and the number of days past due for outstanding balances.

comm key See *communication key.*

communication key In PeopleSoft Enterprise Campus Solutions, a single code for entering a combination of communication category, communication context, communication method, communication direction, and standard letter code. Communication keys (also called *comm keys* or *speed keys*) can be created for background processes as well as for specific users.

compensation object In PeopleSoft Enterprise Incentive Management, a node within a compensation structure. Compensation objects are the building blocks that make up a compensation structure's hierarchical representation.

compensation structure In PeopleSoft Enterprise Incentive Management, a hierarchical relationship of compensation objects that represents the compensation-related relationship between the objects.

condition In PeopleSoft Receivables, occurs when there is a change of status for a customer's account, such as reaching a credit limit or exceeding a user-defined balance due.

configuration parameter catalog Used to configure an external system with PeopleSoft. For example, a configuration parameter catalog might set up configuration and communication parameters for an external server.

configuration plan In PeopleSoft Enterprise Incentive Management, configuration plans hold allocation information for common variables (not incentive rules) and are attached to a node without a participant. Configuration plans are not processed by transactions.

constituents In PeopleSoft Enterprise Campus Solutions, friends, alumni, organizations, foundations, or other entities affiliated with the institution, and about which the institution maintains information. The constituent types delivered with PeopleSoft Enterprise Contributor Relations Solutions are based on those defined by the Council for the Advancement and Support of Education (CASE).

content reference Content references are pointers to content registered in the portal registry. These are typically either URLs or iScripts. Content references fall into three categories: target content, templates, and template pagelets.

context In PeopleCode, determines which buffer fields can be contextually referenced and which is the current row of data on each scroll level when a PeopleCode program is running.

In PeopleSoft Enterprise Campus Solutions, a specific instance of a comment or communication. One or more contexts are assigned to a category, which you link to 3C access groups so that you can assign data-entry or view-only privileges across functions.

In PeopleSoft Enterprise Incentive Management, a mechanism that is used to determine the scope of a processing run. PeopleSoft Enterprise Incentive Management uses three types of context: plan, period, and run-level.

control table Stores information that controls the processing of an application. This type of processing might be consistent throughout an organization, or it might be used only by portions of the organization for more limited sharing of data.

cost profile A combination of a receipt cost method, a cost flow, and a deplete cost method. A profile is associated with a cost book and determines how items in that book are valued, as well as how the material movement of the item is valued for the book.

cost row A cost transaction and amount for a set of ChartFields.

course In PeopleSoft Enterprise Campus Solutions, a course that is offered by a school and that is typically described in a course catalog. A course has a standard syllabus and credit level; however, these may be modified at the class level. Courses can contain multiple components such as lecture, discussion, and lab.

See also *class*.

course share set In PeopleSoft Enterprise Campus Solutions, a tag that defines a set of requirement groups that can share courses. Course share sets are used in PeopleSoft Enterprise Academic Advisement.

current learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's in-progress learning activities and programs.

data acquisition In PeopleSoft Enterprise Incentive Management, the process during which raw business transactions are acquired from external source systems and fed into the operational data store (ODS).

data elements Data elements, at their simplest level, define a subset of data and the rules by which to group them.

For Workforce Analytics, data elements are rules that tell the system what measures to retrieve about your workforce groups.

dataset A data grouping that enables role-based filtering and distribution of data. You can limit the range and quantity of data that is displayed for a user by associating dataset rules with user roles. The result of dataset rules is a set of data that is appropriate for the user's roles.

delivery method In PeopleSoft Enterprise Learning Management, identifies the primary type of delivery method in which a particular learning activity is offered. Also provides

	default values for the learning activity, such as cost and language. This is primarily used to help learners search the catalog for the type of delivery from which they learn best. Because PeopleSoft Enterprise Learning Management is a blended learning system, it does not enforce the delivery method. In PeopleSoft Supply Chain Management, identifies the method by which goods are shipped to their destinations (such as truck, air, rail, and so on). The delivery method is specified when creating shipment schedules.
delivery method type	In PeopleSoft Enterprise Learning Management, identifies how learning activities can be delivered—for example, through online learning, classroom instruction, seminars, books, and so forth—in an organization. The type determines whether the delivery method includes scheduled components.
directory information tree	In PeopleSoft Directory Interface, the representation of a directory's hierarchical structure.
division	In PeopleSoft Enterprise Campus Solutions, the lowest level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a division level, link it to other levels, and set enrollment target numbers for it. See also *population* and *cohort*.
document sequencing	A flexible method that sequentially numbers the financial transactions (for example, bills, purchase orders, invoices, and payments) in the system for statutory reporting and for tracking commercial transaction activity.
dynamic detail tree	A tree that takes its detail values—dynamic details—directly from a table in the database, rather than from a range of values that are entered by the user.
edit table	A table in the database that has its own record definition, such as the Department table. As fields are entered into a PeopleSoft application, they can be validated against an edit table to ensure data integrity throughout the system.
effective date	A method of dating information in PeopleSoft applications. You can predate information to add historical data to your system, or postdate information in order to enter it before it actually goes into effect. By using effective dates, you don't delete values; you enter a new value with a current effective date.
EIM ledger	Abbreviation for *Enterprise Incentive Management ledger.* In PeopleSoft Enterprise Incentive Management, an object to handle incremental result gathering within the scope of a participant. The ledger captures a result set with all of the appropriate traces to the data origin and to the processing steps of which it is a result.
elimination set	In PeopleSoft General Ledger, a related group of intercompany accounts that is processed during consolidations.
entry event	In PeopleSoft General Ledger, Receivables, Payables, Purchasing, and Billing, a business process that generates multiple debits and credits resulting from single transactions to produce standard, supplemental accounting entries.
equitization	In PeopleSoft General Ledger, a business process that enables parent companies to calculate the net income of subsidiaries on a monthly basis and adjust that amount to increase the investment amount and equity income amount before performing consolidations.
equity item limit	In PeopleSoft Enterprise Campus Solutions, the amounts of funds set by the institution to be awarded with discretionary or gift funds. The limit could be reduced by amounts equal to such things as expected family contribution (EFC) or parent contribution. Students are packaged by Equity Item Type Groups and Related Equity Item Types. This limit can be used to assure that similar student populations are packaged equally.

event A predefined point either in the Component Processor flow or in the program flow. As each point is encountered, the event activates each component, triggering any PeopleCode program that is associated with that component and that event. Examples of events are FieldChange, SavePreChange, and RowDelete.

In PeopleSoft Human Resources, also refers to an incident that affects benefits eligibility.

event propagation process In PeopleSoft Sales Incentive Management, a process that determines, through logic, the propagation of an original PeopleSoft Enterprise Incentive Management event and creates a derivative (duplicate) of the original event to be processed by other objects. Sales Incentive Management uses this mechanism to implement splits, roll-ups, and so on. Event propagation determines who receives the credit.

exception In PeopleSoft Receivables, an item that either is a deduction or is in dispute.

exclusive pricing In PeopleSoft Order Management, a type of arbitration plan that is associated with a price rule. Exclusive pricing is used to price sales order transactions.

fact In PeopleSoft applications, facts are numeric data values from fields from a source database as well as an analytic application. A fact can be anything you want to measure your business by, for example, revenue, actual, budget data, or sales numbers. A fact is stored on a fact table.

financial aid term In PeopleSoft Enterprise Campus Solutions, a combination of a period of time that the school determines as an instructional accounting period and an academic career. It is created and defined during the setup process. Only terms eligible for financial aid are set up for each financial aid career.

forecast item A logical entity with a unique set of descriptive demand and forecast data that is used as the basis to forecast demand. You create forecast items for a wide range of uses, but they ultimately represent things that you buy, sell, or use in your organization and for which you require a predictable usage.

fund In PeopleSoft Promotions Management, a budget that can be used to fund promotional activity. There are four funding methods: top down, fixed accrual, rolling accrual, and zero-based accrual.

gap In PeopleSoft Enterprise Campus Solutions, an artificial figure that sets aside an amount of unmet financial aid need that is not funded with Title IV funds. A gap can be used to prevent fully funding any student to conserve funds, or it can be used to preserve unmet financial aid need so that institutional funds can be awarded.

generic process type In PeopleSoft Process Scheduler, process types are identified by a generic process type. For example, the generic process type SQR includes all SQR process types, such as SQR process and SQR report.

gift table In PeopleSoft Enterprise Campus Solutions, a table or so-called *donor pyramid* describing the number and size of gifts that you expect will be needed to successfully complete the campaign in PeopleSoft Contributor Relations. The gift table enables you to estimate the number of donors and prospects that you need at each gift level to reach the campaign goal.

GL business unit Abbreviation for *general ledger business unit.* A unit in an organization that is an independent entity for accounting purposes. It maintains its own set of accounting books.

See also *business unit*.

GL entry template Abbreviation for *general ledger entry template.* In PeopleSoft Enterprise Campus Solutions, a template that defines how a particular item is sent to the general ledger. An item-type maps to the general ledger, and the GL entry template can involve multiple general ledger accounts. The entry to the general ledger is further controlled

by high-level flags that control the summarization and the type of accounting—that is, accrual or cash.

GL Interface process Abbreviation for *General Ledger Interface process.* In PeopleSoft Enterprise Campus Solutions, a process that is used to send transactions from PeopleSoft Enterprise Student Financials to the general ledger. Item types are mapped to specific general ledger accounts, enabling transactions to move to the general ledger when the GL Interface process is run.

group In PeopleSoft Billing and Receivables, a posting entity that comprises one or more transactions (items, deposits, payments, transfers, matches, or write-offs).

In PeopleSoft Human Resources Management and Supply Chain Management, any set of records that are associated under a single name or variable to run calculations in PeopleSoft business processes. In PeopleSoft Time and Labor, for example, employees are placed in groups for time reporting purposes.

incentive object In PeopleSoft Enterprise Incentive Management, the incentive-related objects that define and support the PeopleSoft Enterprise Incentive Management calculation process and results, such as plan templates, plans, results data, user interaction objects, and so on.

incentive rule In PeopleSoft Sales Incentive Management, the commands that act on transactions and turn them into compensation. A rule is one part in the process of turning a transaction into compensation.

incur In PeopleSoft Promotions Management, to become liable for a promotional payment. In other words, you owe that amount to a customer for promotional activities.

initiative In PeopleSoft Enterprise Campus Solutions, the basis from which all advancement plans are executed. It is an organized effort targeting a specific constituency, and it can occur over a specified period of time with specific purposes and goals. An initiative can be a campaign, an event, an organized volunteer effort, a membership drive, or any other type of effort defined by the institution. Initiatives can be multipart, and they can be related to other initiatives. This enables you to track individual parts of an initiative, as well as entire initiatives.

inquiry access In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user only to view data.

See also *update access.*

institution In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.

item In PeopleSoft Inventory, a tangible commodity that is stored in a business unit (shipped from a warehouse).

In PeopleSoft Demand Planning, Inventory Policy Planning, and Supply Planning, a noninventory item that is designated as being used for planning purposes only. It can represent a family or group of inventory items. It can have a planning bill of material (BOM) or planning routing, and it can exist as a component on a planning BOM. A planning item cannot be specified on a production or engineering BOM or routing, and it cannot be used as a component in a production. The quantity on hand will never be maintained.

In PeopleSoft Receivables, an individual receivable. An item can be an invoice, a credit memo, a debit memo, a write-off, or an adjustment.

item shuffle In PeopleSoft Enterprise Campus Solutions, a process that enables you to change a payment allocation without having to reverse the payment.

joint communication In PeopleSoft Enterprise Campus Solutions, one letter that is addressed jointly to two people. For example, a letter might be addressed to both Mr. Sudhir Awat and Ms. Samantha Mortelli. A relationship must be established between the two individuals in the database, and at least one of the individuals must have an ID in the database.

keyword In PeopleSoft Enterprise Campus Solutions, a term that you link to particular elements within PeopleSoft Student Financials, Financial Aid, and Contributor Relations. You can use keywords as search criteria that enable you to locate specific records in a search dialog box.

KPI An abbreviation for *key performance indicator.* A high-level measurement of how well an organization is doing in achieving critical success factors. This defines the data value or calculation upon which an assessment is determined.

LDIF file Abbreviation for *Lightweight Directory Access Protocol (LDAP) Data Interchange Format file.* Contains discrepancies between PeopleSoft data and directory data.

learner group In PeopleSoft Enterprise Learning Management, a group of learners who are linked to the same learning environment. Members of the learner group can share the same attributes, such as the same department or job code. Learner groups are used to control access to and enrollment in learning activities and programs. They are also used to perform group enrollments and mass enrollments in the back office.

learning components In PeopleSoft Enterprise Learning Management, the foundational building blocks of learning activities. PeopleSoft Enterprise Learning Management supports six basic types of learning components: web-based, session, webcast, test, survey, and assignment. One or more of these learning component types compose a single learning activity.

learning environment In PeopleSoft Enterprise Learning Management, identifies a set of categories and catalog items that can be made available to learner groups. Also defines the default values that are assigned to the learning activities and programs that are created within a particular learning environment. Learning environments provide a way to partition the catalog so that learners see only those items that are relevant to them.

learning history In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's completed learning activities and programs.

ledger mapping You use ledger mapping to relate expense data from general ledger accounts to resource objects. Multiple ledger line items can be mapped to one or more resource IDs. You can also use ledger mapping to map dollar amounts (referred to as *rates*) to business units. You can map the amounts in two different ways: an actual amount that represents actual costs of the accounting period, or a budgeted amount that can be used to calculate the capacity rates as well as budgeted model results. In PeopleSoft Enterprise Warehouse, you can map general ledger accounts to the EW Ledger table.

library section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan (or template) and that is available for other plans to share. Changes to a library section are reflected in all plans that use it.

linked section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan template but appears in a plan. Changes to linked sections propagate to plans using that section.

linked variable In PeopleSoft Enterprise Incentive Management, a variable that is defined and maintained in a plan template and that also appears in a plan. Changes to linked variables propagate to plans using that variable.

LMS Abbreviation for *learning management system.* In PeopleSoft Enterprise Campus Solutions, LMS is a PeopleSoft Student Records feature that provides a common set of interoperability standards that enable the sharing of instructional content and data between learning and administrative environments.

load In PeopleSoft Inventory, identifies a group of goods that are shipped together. Load management is a feature of PeopleSoft Inventory that is used to track the weight, the volume, and the destination of a shipment.

local functionality In PeopleSoft HRMS, the set of information that is available for a specific country. You can access this information when you click the appropriate country flag in the global window, or when you access it by a local country menu.

location Locations enable you to indicate the different types of addresses—for a company, for example, one address to receive bills, another for shipping, a third for postal deliveries, and a separate street address. Each address has a different location number. The primary location—indicated by a *1*—is the address you use most often and may be different from the main address.

logistical task In PeopleSoft Services Procurement, an administrative task that is related to hiring a service provider. Logistical tasks are linked to the service type on the work order so that different types of services can have different logistical tasks. Logistical tasks include both preapproval tasks (such as assigning a new badge or ordering a new laptop) and postapproval tasks (such as scheduling orientation or setting up the service provider email). The logistical tasks can be mandatory or optional. Mandatory preapproval tasks must be completed before the work order is approved. Mandatory postapproval tasks, on the other hand, must be completed before a work order is released to a service provider.

market template In PeopleSoft Enterprise Incentive Management, additional functionality that is specific to a given market or industry and is built on top of a product category.

mass change In PeopleSoft Enterprise Campus Solutions, mass change is a SQL generator that can be used to create specialized functionality. Using mass change, you can set up a series of Insert, Update, or Delete SQL statements to perform business functions that are specific to the institution.

See also *3C engine*.

match group In PeopleSoft Receivables, a group of receivables items and matching offset items. The system creates match groups by using user-defined matching criteria for selected field values.

MCF server Abbreviation for *PeopleSoft MultiChannel Framework server.* Comprises the universal queue server and the MCF log server. Both processes are started when *MCF Servers* is selected in an application server domain configuration.

merchandising activity In PeopleSoft Promotions Management, a specific discount type that is associated with a trade promotion (such as off-invoice, billback or rebate, or lump-sum payment) that defines the performance that is required to receive the discount. In the industry, you may know this as an offer, a discount, a merchandising event, an event, or a tactic.

meta-SQL Meta-SQL constructs expand into platform-specific Structured Query Language (SQL) substrings. They are used in functions that pass SQL strings, such as in SQL objects, the SQLExec function, and PeopleSoft Application Engine programs.

metastring Metastrings are special expressions included in SQL string literals. The metastrings, prefixed with a percent (%) symbol, are included directly in the string literals. They expand at run time into an appropriate substring for the current database platform.

multibook In PeopleSoft General Ledger, multiple ledgers having multiple-base currencies that are defined for a business unit, with the option to post a single transaction to all base currencies (all ledgers) or to only one of those base currencies (ledgers).

multicurrency The ability to process transactions in a currency other than the business unit's base currency.

national allowance In PeopleSoft Promotions Management, a promotion at the corporate level that is funded by nondiscretionary dollars. In the industry, you may know this as a national promotion, a corporate promotion, or a corporate discount.

need In PeopleSoft Enterprise Campus Solutions, the difference between the cost of attendance (COA) and the expected family contribution (EFC). It is the gap between the cost of attending the school and the student's resources. The financial aid package is based on the amount of financial need. The process of determining a student's need is called *need analysis.*

node-oriented tree A tree that is based on a detail structure, but the detail values are not used.

pagelet Each block of content on the home page is called a pagelet. These pagelets display summary information within a small rectangular area on the page. The pagelet provide users with a snapshot of their most relevant PeopleSoft and non-PeopleSoft content.

participant In PeopleSoft Enterprise Incentive Management, participants are recipients of the incentive compensation calculation process.

participant object Each participant object may be related to one or more compensation objects.

See also *compensation object.*

partner A company that supplies products or services that are resold or purchased by the enterprise.

pay cycle In PeopleSoft Payables, a set of rules that define the criteria by which it should select scheduled payments for payment creation.

payment shuffle In PeopleSoft Enterprise Campus Solutions, a process allowing payments that have been previously posted to a student's account to be automatically reapplied when a higher priority payment is posted or the payment allocation definition is changed.

pending item In PeopleSoft Receivables, an individual receivable (such as an invoice, a credit memo, or a write-off) that has been entered in or created by the system, but hasn't been posted.

PeopleCode PeopleCode is a proprietary language, executed by the PeopleSoft application processor. PeopleCode generates results based upon existing data or user actions. By using business interlink objects, external services are available to all PeopleSoft applications wherever PeopleCode can be executed.

PeopleCode event An action that a user takes upon an object, usually a record field, that is referenced within a PeopleSoft page.

PeopleSoft Internet Architecture The fundamental architecture on which PeopleSoft 8 applications are constructed, consisting of a relational database management system (RDBMS), an application server, a web server, and a browser.

performance measurement In PeopleSoft Enterprise Incentive Management, a variable used to store data (similar to an aggregator, but without a predefined formula) within the scope of an incentive plan. Performance measures are associated with a plan calendar, territory, and participant. Performance measurements are used for quota calculation and reporting.

period context In PeopleSoft Enterprise Incentive Management, because a participant typically uses the same compensation plan for multiple periods, the period context associates a plan context with a specific calendar period and fiscal year. The period context references the associated plan context, thus forming a chain. Each plan context has a corresponding set of period contexts.

person of interest A person about whom the organization maintains information but who is not part of the workforce.

personal portfolio In PeopleSoft Enterprise Campus Solutions, the user-accessible menu item that contains an individual's name, address, telephone number, and other personal information.

plan In PeopleSoft Sales Incentive Management, a collection of allocation rules, variables, steps, sections, and incentive rules that instruct the PeopleSoft Enterprise Incentive Management engine in how to process transactions.

plan context In PeopleSoft Enterprise Incentive Management, correlates a participant with the compensation plan and node to which the participant is assigned, enabling the PeopleSoft Enterprise Incentive Management system to find anything that is associated with the node and that is required to perform compensation processing. Each participant, node, and plan combination represents a unique plan context—if three participants are on a compensation structure, each has a different plan context. Configuration plans are identified by plan contexts and are associated with the participants that refer to them.

plan template In PeopleSoft Enterprise Incentive Management, the base from which a plan is created. A plan template contains common sections and variables that are inherited by all plans that are created from the template. A template may contain steps and sections that are not visible in the plan definition.

planned learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's planned learning activities and programs.

planning instance In PeopleSoft Supply Planning, a set of data (business units, items, supplies, and demands) constituting the inputs and outputs of a supply plan.

population In PeopleSoft Enterprise Campus Solutions, the middle level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a population level, link it to other levels, and set enrollment target numbers for it.

See also *division* and *cohort*.

portal registry In PeopleSoft applications, the portal registry is a tree-like structure in which content references are organized, classified, and registered. It is a central repository that defines both the structure and content of a portal through a hierarchical, tree-like structure of folders useful for organizing and securing content references.

price list In PeopleSoft Enterprise Pricer, enables you to select products and conditions for which the price list applies to a transaction. During a transaction, the system either determines the product price based on the predefined search hierarchy for the transaction or uses the product's lowest price on any associated, active price lists. This price is used as the basis for any further discounts and surcharges.

price rule In PeopleSoft Enterprise Pricer, defines the conditions that must be met for adjustments to be applied to the base price. Multiple rules can apply when conditions of each rule are met.

price rule condition In PeopleSoft Enterprise Pricer, selects the price-by fields, the values for the price-by fields, and the operator that determines how the price-by fields are related to the transaction.

price rule key In PeopleSoft Enterprise Pricer, defines the fields that are available to define price rule conditions (which are used to match a transaction) on the price rule.

primacy number In PeopleSoft Enterprise Campus Solutions, a number that the system uses to prioritize financial aid applications when students are enrolled in multiple academic careers and academic programs at the same time. The Consolidate Academic Statistics process uses the primacy number indicated for both the career and program at the institutional level to determine a student's primary career and program. The system also uses the

	number to determine the primary student attribute value that is used when you extract data to report on cohorts. The lowest number takes precedence.
primary name type	In PeopleSoft Enterprise Campus Solutions, the name type that is used to link the name stored at the highest level within the system to the lower-level set of names that an individual provides.
process category	In PeopleSoft Process Scheduler, processes that are grouped for server load balancing and prioritization.
process group	In PeopleSoft Financials, a group of application processes (performed in a defined order) that users can initiate in real time, directly from a transaction entry page.
process definition	Process definitions define each run request.
process instance	A unique number that identifies each process request. This value is automatically incremented and assigned to each requested process when the process is submitted to run.
process job	You can link process definitions into a job request and process each request serially or in parallel. You can also initiate subsequent processes based on the return code from each prior request.
process request	A single run request, such as a Structured Query Report (SQR), a COBOL or Application Engine program, or a Crystal report that you run through PeopleSoft Process Scheduler.
process run control	A PeopleTools variable used to retain PeopleSoft Process Scheduler values needed at runtime for all requests that reference a run control ID. Do not confuse these with application run controls, which may be defined with the same run control ID, but only contain information specific to a given application process request.
product category	In PeopleSoft Enterprise Incentive Management, indicates an application in the Enterprise Incentive Management suite of products. Each transaction in the PeopleSoft Enterprise Incentive Management system is associated with a product category.
programs	In PeopleSoft Enterprise Learning Management, a high-level grouping that guides the learner along a specific learning path through sections of catalog items. PeopleSoft Enterprise Learning Systems provides two types of programs—curricula and certifications.
progress log	In PeopleSoft Services Procurement, tracks deliverable-based projects. This is similar to the time sheet in function and process. The service provider contact uses the progress log to record and submit progress on deliverables. The progress can be logged by the activity that is performed, by the percentage of work that is completed, or by the completion of milestone activities that are defined for the project.
project transaction	In PeopleSoft Project Costing, an individual transaction line that represents a cost, time, budget, or other transaction row.
promotion	In PeopleSoft Promotions Management, a trade promotion, which is typically funded from trade dollars and used by consumer products manufacturers to increase sales volume.
prospects	In PeopleSoft Enterprise Campus Solutions, students who are interested in applying to the institution. In PeopleSoft Enterprise Contributor Relations, individuals and organizations that are most likely to make substantial financial commitments or other types of commitments to the institution.
publishing	In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.

rating components In PeopleSoft Enterprise Campus Solutions, variables used with the Equation Editor to retrieve specified populations.

record group A set of logically and functionally related control tables and views. Record groups help enable TableSet sharing, which eliminates redundant data entry. Record groups ensure that TableSet sharing is applied consistently across all related tables and views.

record input VAT flag Abbreviation for *record input value-added tax flag*. Within PeopleSoft Purchasing, Payables, and General Ledger, this flag indicates that you are recording input VAT on the transaction. This flag, in conjunction with the record output VAT flag, is used to determine the accounting entries created for a transaction and to determine how a transaction is reported on the VAT return. For all cases within Purchasing and Payables where VAT information is tracked on a transaction, this flag is set to Yes. This flag is not used in PeopleSoft Order Management, Billing, or Receivables, where it is assumed that you are always recording only output VAT, or in PeopleSoft Expenses, where it is assumed that you are always recording only input VAT.

record output VAT flag Abbreviation for *record output value-added tax flag.*

See *record input VAT flag.*

recname The name of a record that is used to determine the associated field to match a value or set of values.

recognition In PeopleSoft Enterprise Campus Solutions, the recognition type indicates whether the PeopleSoft Enterprise Contributor Relations donor is the primary donor of a commitment or shares the credit for a donation. Primary donors receive hard credit that must total 100 percent. Donors that share the credit are given soft credit. Institutions can also define other share recognition-type values such as memo credit or vehicle credit.

reference data In PeopleSoft Sales Incentive Management, system objects that represent the sales organization, such as territories, participants, products, customers, channels, and so on.

reference object In PeopleSoft Enterprise Incentive Management, this dimension-type object further defines the business. Reference objects can have their own hierarchy (for example, product tree, customer tree, industry tree, and geography tree).

reference transaction In commitment control, a reference transaction is a source transaction that is referenced by a higher-level (and usually later) source transaction, in order to automatically reverse all or part of the referenced transaction's budget-checked amount. This avoids duplicate postings during the sequential entry of the transaction at different commitment levels. For example, the amount of an encumbrance transaction (such as a purchase order) will, when checked and recorded against a budget, cause the system to concurrently reference and relieve all or part of the amount of a corresponding pre-encumbrance transaction, such as a purchase requisition.

regional sourcing In PeopleSoft Purchasing, provides the infrastructure to maintain, display, and select an appropriate vendor and vendor pricing structure that is based on a regional sourcing model where the multiple ship to locations are grouped. Sourcing may occur at a level higher than the ship to location.

relationship object In PeopleSoft Enterprise Incentive Management, these objects further define a compensation structure to resolve transactions by establishing associations between compensation objects and business objects.

remote data source data Data that is extracted from a separate database and migrated into the local database.

REN server Abbreviation for *real-time event notification server* in PeopleSoft MultiChannel Framework.

requester In PeopleSoft eSettlements, an individual who requests goods or services and whose ID appears on the various procurement pages that reference purchase orders.

reversal indicator In PeopleSoft Enterprise Campus Solutions, an indicator that denotes when a particular payment has been reversed, usually because of insufficient funds.

role Describes how people fit into PeopleSoft Workflow. A role is a class of users who perform the same type of work, such as clerks or managers. Your business rules typically specify what user role needs to do an activity.

role user A PeopleSoft Workflow user. A person's role user ID serves much the same purpose as a user ID does in other parts of the system. PeopleSoft Workflow uses role user IDs to determine how to route worklist items to users (through an email address, for example) and to track the roles that users play in the workflow. Role users do not need PeopleSoft user IDs.

roll up In a tree, to roll up is to total sums based on the information hierarchy.

run control A run control is a type of online page that is used to begin a process, such as the batch processing of a payroll run. Run control pages generally start a program that manipulates data.

run control ID A unique ID to associate each user with his or her own run control table entries.

run-level context In PeopleSoft Enterprise Incentive Management, associates a particular run (and batch ID) with a period context and plan context. Every plan context that participates in a run has a separate run-level context. Because a run cannot span periods, only one run-level context is associated with each plan context.

search query You use this set of objects to pass a query string and operators to the search engine. The search index returns a set of matching results with keys to the source documents.

search/match In PeopleSoft Enterprise Campus Solutions and PeopleSoft Enterprise Human Resources Management Solutions, a feature that enables you to search for and identify duplicate records in the database.

seasonal address In PeopleSoft Enterprise Campus Solutions, an address that recurs for the same length of time at the same time of year each year until adjusted or deleted.

section In PeopleSoft Enterprise Incentive Management, a collection of incentive rules that operate on transactions of a specific type. Sections enable plans to be segmented to process logical events in different sections.

security event In commitment control, security events trigger security authorization checking, such as budget entries, transfers, and adjustments; exception overrides and notifications; and inquiries.

serial genealogy In PeopleSoft Manufacturing, the ability to track the composition of a specific, serial-controlled item.

serial in production In PeopleSoft Manufacturing, enables the tracing of serial information for manufactured items. This is maintained in the Item Master record.

service impact In PeopleSoft Enterprise Campus Solutions, the resulting action triggered by a service indicator. For example, a service indicator that reflects nonpayment of account balances by a student might result in a service impact that prohibits registration for classes.

service indicator In PeopleSoft Enterprise Campus Solutions, indicates services that may be either withheld or provided to an individual. Negative service indicators indicate holds that prevent the individual from receiving specified services, such as check-cashing privileges or registration for classes. Positive service indicators designate special services that are provided to the individual, such as front-of-line service or special services for disabled students.

session In PeopleSoft Enterprise Campus Solutions, time elements that subdivide a term into multiple time periods during which classes are offered. In PeopleSoft Contributor Relations, a session is the means of validating gift, pledge, membership, or adjustment data entry . It controls access to the data entered by a specific user ID. Sessions are balanced, queued, and then posted to the institution's financial system. Sessions must be posted to enter a matching gift or pledge payment, to make an adjustment, or to process giving clubs or acknowledgements.

In PeopleSoft Enterprise Learning Management, a single meeting day of an activity (that is, the period of time between start and finish times within a day). The session stores the specific date, location, meeting time, and instructor. Sessions are used for scheduled training.

session template In PeopleSoft Enterprise Learning Management, enables you to set up common activity characteristics that may be reused while scheduling a PeopleSoft Enterprise Learning Management activity—characteristics such as days of the week, start and end times, facility and room assignments, instructors, and equipment. A session pattern template can be attached to an activity that is being scheduled. Attaching a template to an activity causes all of the default template information to populate the activity session pattern.

setup relationship In PeopleSoft Enterprise Incentive Management, a relationship object type that associates a configuration plan with any structure node.

share driver expression In PeopleSoft Business Planning, a named planning method similar to a driver expression, but which you can set up globally for shared use within a single planning application or to be shared between multiple planning applications through PeopleSoft Enterprise Warehouse.

single signon With single signon, users can, after being authenticated by a PeopleSoft application server, access a second PeopleSoft application server without entering a user ID or password.

source key process In PeopleSoft Enterprise Campus Solutions, a process that relates a particular transaction to the source of the charge or financial aid. On selected pages, you can drill down into particular charges.

source transaction In commitment control, any transaction generated in a PeopleSoft or third-party application that is integrated with commitment control and which can be checked against commitment control budgets. For example, a pre-encumbrance, encumbrance, expenditure, recognized revenue, or collected revenue transaction.

speed key See *communication key.*

SpeedChart A user-defined shorthand key that designates several ChartKeys to be used for voucher entry. Percentages can optionally be related to each ChartKey in a SpeedChart definition.

SpeedType A code representing a combination of ChartField values. SpeedTypes simplify the entry of ChartFields commonly used together.

staging A method of consolidating selected partner offerings with the offerings from the enterprise's other partners.

standard letter code In PeopleSoft Enterprise Campus Solutions, a standard letter code used to identify each letter template available for use in mail merge functions. Every letter generated in the system must have a standard letter code identification.

statutory account Account required by a regulatory authority for recording and reporting financial results. In PeopleSoft, this is equivalent to the Alternate Account (ALTACCT) ChartField.

step In PeopleSoft Sales Incentive Management, a collection of sections in a plan. Each step corresponds to a step in the job run.

storage level In PeopleSoft Inventory, identifies the level of a material storage location. Material storage locations are made up of a business unit, a storage area, and a storage level. You can set up to four storage levels.

subcustomer qualifier A value that groups customers into a division for which you can generate detailed history, aging, events, and profiles.

Summary ChartField You use summary ChartFields to create summary ledgers that roll up detail amounts based on specific detail values or on selected tree nodes. When detail values are summarized using tree nodes, summary ChartFields must be used in the summary ledger data record to accommodate the maximum length of a node name (20 characters).

summary ledger An accounting feature used primarily in allocations, inquiries, and PS/nVision reporting to store combined account balances from detail ledgers. Summary ledgers increase speed and efficiency of reporting by eliminating the need to summarize detail ledger balances each time a report is requested. Instead, detail balances are summarized in a background process according to user-specified criteria and stored on summary ledgers. The summary ledgers are then accessed directly for reporting.

summary time period In PeopleSoft Business Planning, any time period (other than a base time period) that is an aggregate of other time periods, including other summary time periods and base time periods, such as quarter and year total.

summary tree A tree used to roll up accounts for each type of report in summary ledgers. Summary trees enable you to define trees on trees. In a summary tree, the detail values are really nodes on a detail tree or another summary tree (known as the *basis* tree). A summary tree structure specifies the details on which the summary trees are to be built.

syndicate To distribute a production version of the enterprise catalog to partners.

system function In PeopleSoft Receivables, an activity that defines how the system generates accounting entries for the general ledger.

TableSet A means of sharing similar sets of values in control tables, where the actual data values are different but the structure of the tables is the same.

TableSet sharing Shared data that is stored in many tables that are based on the same TableSets. Tables that use TableSet sharing contain the SETID field as an additional key or unique identifier.

target currency The value of the entry currency or currencies converted to a single currency for budget viewing and inquiry purposes.

tax authority In PeopleSoft Enterprise Campus Solutions, a user-defined element that combines a description and percentage of a tax with an account type, an item type, and a service impact.

template A template is HTML code associated with a web page. It defines the layout of the page and also where to get HTML for each part of the page. In PeopleSoft, you use templates to build a page by combining HTML from a number of sources. For a PeopleSoft portal, all templates must be registered in the portal registry, and each content reference must be assigned a template.

territory In PeopleSoft Sales Incentive Management, hierarchical relationships of business objects, including regions, products, customers, industries, and participants.

3C engine Abbreviation for *Communications, Checklists, and Comments engine.* In PeopleSoft Enterprise Campus Solutions, the 3C engine enables you to automate business processes that involve additions, deletions, and updates to communications, checklists,

	and comments. You define events and triggers to engage the engine, which runs the mass change and processes the 3C records (for individuals or organizations) immediately and automatically from within business processes.
3C group	Abbreviation for *Communications, Checklists, and Comments group.* In PeopleSoft Enterprise Campus Solutions, a method of assigning or restricting access privileges. A 3C group enables you to group specific communication categories, checklist codes, and comment categories. You can then assign the group inquiry-only access or update access, as appropriate.
TimeSpan	A relative period, such as year-to-date or current period, that can be used in various PeopleSoft General Ledger functions and reports when a rolling time frame, rather than a specific date, is required. TimeSpans can also be used with flexible formulas in PeopleSoft Projects.
trace usage	In PeopleSoft Manufacturing, enables the control of which components will be traced during the manufacturing process. Serial- and lot-controlled components can be traced. This is maintained in the Item Master record.
transaction allocation	In PeopleSoft Enterprise Incentive Management, the process of identifying the owner of a transaction. When a raw transaction from a batch is allocated to a plan context, the transaction is duplicated in the PeopleSoft Enterprise Incentive Management transaction tables.
transaction state	In PeopleSoft Enterprise Incentive Management, a value assigned by an incentive rule to a transaction. Transaction states enable sections to process only transactions that are at a specific stage in system processing. After being successfully processed, transactions may be promoted to the next transaction state and "picked up" by a different section for further processing.
Translate table	A system edit table that stores codes and translate values for the miscellaneous fields in the database that do not warrant individual edit tables of their own.
tree	The graphical hierarchy in PeopleSoft systems that displays the relationship between all accounting units (for example, corporate divisions, projects, reporting groups, account numbers) and determines roll-up hierarchies.
tuition lock	In PeopleSoft Enterprise Campus Solutions, a feature in the Tuition Calculation process that enables you to specify a point in a term after which students are charged a minimum (or *locked*) fee amount. Students are charged the locked fee amount even if they later drop classes and take less than the normal load level for that tuition charge.
unclaimed transaction	In PeopleSoft Enterprise Incentive Management, a transaction that is not claimed by a node or participant after the allocation process has completed, usually due to missing or incomplete data. Unclaimed transactions may be manually assigned to the appropriate node or participant by a compensation administrator.
universal navigation header	Every PeopleSoft portal includes the universal navigation header, intended to appear at the top of every page as long as the user is signed on to the portal. In addition to providing access to the standard navigation buttons (like Home, Favorites, and signoff) the universal navigation header can also display a welcome message for each user.
update access	In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user to edit and update data. See also *inquiry access*.
user interaction object	In PeopleSoft Sales Incentive Management, used to define the reporting components and reports that a participant can access in his or her context. All Sales Incentive Management user interface objects and reports are registered as user interaction objects. User interaction objects can be linked to a compensation structure node through a compensation relationship object (individually or as groups).

variable In PeopleSoft Sales Incentive Management, the intermediate results of calculations. Variables hold the calculation results and are then inputs to other calculations. Variables can be plan variables that persist beyond the run of an engine or local variables that exist only during the processing of a section.

VAT exception Abbreviation for *value-added tax exception.* A temporary or permanent exemption from paying VAT that is granted to an organization. This terms refers to both VAT exoneration and VAT suspension.

VAT exempt Abbreviation for *value-added tax exempt.* Describes goods and services that are not subject to VAT. Organizations that supply exempt goods or services are unable to recover the related input VAT. This is also referred to as exempt without recovery.

VAT exoneration Abbreviation for *value-added tax exoneration.* An organization that has been granted a permanent exemption from paying VAT due to the nature of that organization.

VAT suspension Abbreviation for *value-added tax suspension.* An organization that has been granted a temporary exemption from paying VAT.

warehouse A PeopleSoft data warehouse that consists of predefined ETL maps, data warehouse tools, and DataMart definitions.

work order In PeopleSoft Services Procurement, enables an enterprise to create resource-based and deliverable-based transactions that specify the basic terms and conditions for hiring a specific service provider. When a service provider is hired, the service provider logs time or progress against the work order.

worker A person who is part of the workforce; an employee or a contingent worker.

workset A group of people and organizations that are linked together as a set. You can use worksets to simultaneously retrieve the data for a group of people and organizations and work with the information on a single page.

worksheet A way of presenting data through a PeopleSoft Business Analysis Modeler interface that enables users to do in-depth analysis using pivoting tables, charts, notes, and history information.

worklist The automated to-do list that PeopleSoft Workflow creates. From the worklist, you can directly access the pages you need to perform the next action, and then return to the worklist for another item.

XML schema An XML definition that standardizes the representation of application messages, component interfaces, or business interlinks.

yield by operation In PeopleSoft Manufacturing, the ability to plan the loss of a manufactured item on an operation-by-operation basis.

zero-rated VAT Abbreviation for *zero-rated value-added tax.* A VAT transaction with a VAT code that has a tax percent of zero. Used to track taxable VAT activity where no actual VAT amount is charged. Organizations that supply zero-rated goods and services can still recover the related input VAT. This is also referred to as exempt with recovery.

Index

P

R

S

T

V

W

X