

Oracle® Process Manufacturing

Inventory API User's Guide

Release 11*i*

June 2002

Part No. A82921-03

Oracle Process Manufacturing Inventory API User's Guide, Release 11i

Part No. A82921-03

Copyright © 2002, Oracle Corporation. All rights reserved.

Primary Author: Michele-Andrea Fields

Contributors: Mark Holst, Barbara Mercurio, Joe DiIorio, Anthony Cataldo, Jatinder Gogna

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Audience for This Guide	ix
How To Use This Guide	ix
Documentation Accessibility	ix
Other Information Sources	x
Online Documentation.....	x
Related User’s Guides.....	xi
Guides Related to All Products	xi
User Guides Related to This Product	xi
Installation and System Administration	xvi
Other Implementation Documentation.....	xvii
Training and Support.....	xix
Do Not Use Database Tools to Modify Oracle Applications Data	xix
About Oracle	xx
Your Feedback	xx

1 OPM Inventory APIs

Introducing the OPM Inventory APIs	1-2
OPM Inventory API Features	1-3
OPM Inventory API Support Policy	1-3
Technical Requirements	1-3
Input Data Sources	1-4
Wrapper Function	1-4
Stored Procedure	1-5
Basic Business Needs	1-5
Major Features	1-6

Item Create API.....	1-6
Item Lot/Sublot Conversion API.....	1-6
OPM Inventory Quantities API.....	1-6
Lot Create API.....	1-6
API Architecture.....	1-8
API Package Details.....	1-8
OPM Inventory API Bill of Materials.....	1-10

2 OPM Inventory API Usage

Item Create API Wrapper - Business Function.....	2-2
Wrapper Function - Input Data Sources.....	2-2
Stored Procedures Overview.....	2-2
Stored Procedure Execution.....	2-3
Calling the API Interface Code.....	2-4
Item Create API Wrapper - Input Structure.....	2-5
Item Create API Wrapper - ASCII Flat File Layout.....	2-5
Log Files.....	2-5
Calling the API Code - Example.....	2-5
Item Create API Wrapper - Code Example.....	2-6

3 Technical Overview

Item Create.....	3-2
Structure for Item Create Public APIs.....	3-2
Item Lot/Sublot Conversion.....	3-3
Structure for Item Lot/Sublot Conversion Public APIs.....	3-3
Inventory Quantities.....	3-4
Structure for Inventory Quantities Public APIs.....	3-4
Lot Create.....	3-5
Structure for Lot Create Public APIs.....	3-5
Standard Parameters.....	3-6
Value-ID Conversion.....	3-7

4 Business Objects

Item Create.....	4-2
-------------------------	------------

Parameters and Interface.....	4-2
Item Lot/Sublot Conversion	4-9
Item Lot/Sublot Conversion API - Parameters.....	4-9
Inventory Quantities	4-11
Inventory Quantities - Parameters.....	4-11
Lot Create	4-14
Lot Create API - Parameters	4-14

A Messages and Errors

Handling Messages	A-1
Interpreting Error Conditions	A-3
Understanding Error Messages.....	A-3
File Error Messages	A-8

B How to Get Your OPM Inventory APIs Running

Send Us Your Comments

Oracle Process Manufacturing Inventory API User's Guide, Release 11i

Part No. A82921-03

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- FAX: 650-506-7200 Attn: Oracle Process Manufacturing
- Postal service:
Oracle Corporation
Oracle Process Manufacturing
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.
- Electronic mail message to appsdoc_us@oracle.com

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Audience for This Guide

Welcome to Release 11i of the *Oracle Process Manufacturing Inventory API User's Guide*.

This guide assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Oracle Process Manufacturing

If you have never used *Oracle Process Manufacturing*, Oracle suggests you attend one or more of the *Oracle Process Manufacturing* training classes available through Oracle University.

- The Oracle Applications graphical user interface.

To learn more about the Oracle Applications graphical user interface, read the *Oracle Applications User's Guide*.

See Other Information Sources for more information about Oracle Applications product information.

How To Use This Guide

This guide contains the information you need to understand and use *Oracle Process Manufacturing*.

- Chapter 1 describes how APIs are used, the basic business need of APIs, and the different OPM Inventory APIs offered.
- Chapter 2 describes how to use the OPM Inventory APIs.
- Chapter 3 describes the technical aspect of the APIs.
- Chapter 4 describes the business objects for each API.
- Appendix A describes messages and error codes.
- Appendix B provides a useful guide and examples for using the APIs.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our

documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Other Information Sources

You can choose from many sources of information, including online documentation, training, and support services, to increase your knowledge and understanding of *Oracle Process Manufacturing*.

If this guide refers you to other Oracle Applications documentation, use only the Release 11*i* versions of those guides.

Online Documentation

All Oracle Applications documentation is available online (HTML or PDF).

- **Online Help** - The new features section in the HTML help describes new features in 11*i*. This information is updated for each new release of *Oracle Process Manufacturing*. The new features section also includes information about any features that were not yet available when this guide was printed. For example, if your administrator has installed software from a mini-packs an upgrade, this document describes the new features. Online help patches are available on MetaLink.
- **11*i* Features Matrix** - This document lists new features available by patch and identifies any associated new documentation. The new features matrix document is available on MetaLink.

- **Readme File** - Refer to the readme file for patches that you have installed to learn about new documentation or documentation patches that you can download.

Related User's Guides

Oracle Process Manufacturing shares business and setup information with other Oracle Applications products. Therefore, you may want to refer to other user's guides when you set up and use *Oracle Process Manufacturing*.

You can read the guides online by choosing Library from the expandable menu on your HTML help window, by reading from the Oracle Applications Document Library CD included in your media pack, or by using a Web browser with a URL that your system administrator provides.

If you require printed guides, you can purchase them from the Oracle Store at <http://oraclestore.oracle.com>.

Guides Related to All Products

Oracle Applications User's Guide

This guide explains how to enter data, query, run reports, and navigate using the graphical user interface (GUI) available with this release of *Oracle Process Manufacturing* (and any other Oracle Applications products). This guide also includes information on setting user profiles, as well as running and reviewing reports and concurrent processes.

You can access this user's guide online by choosing "Getting Started with Oracle Applications" from any Oracle Applications help file.

User Guides Related to This Product

Accounting Setup User's Guide

The OPM Accounting Setup application is where users set up global accounting attributes about the way financial data will be collected by OPM. These attributes include such things as account keys, financial calendars, and account segments. Since OPM is closely integrated with Oracle General Ledger (GL), much of the attributes are defined in the Oracle GL instead of OPM, and therefore, the windows are display only within OPM. The *Oracle Process Manufacturing Accounting Setup User's Guide* describes how to setup and use this application.

Cost Management User's Guide

The OPM Cost Management application is used by cost accountants to capture and review the manufacturing costs incurred in their process manufacturing businesses. The *Oracle Process Manufacturing Cost Management User's Guide* describes how to setup and use this application.

Manufacturing Accounting Controller User's Guide

The Manufacturing Accounting Controller application is where users define the impact of manufacturing events on financials. For example, event RCPT (Inventory Receipts) results in a debit to inventory, a credit to accrued accounts payable, a debit or a credit to purchase price variance, etc. These impacts are predefined in the Manufacturing Accounting Controller application so users may begin using OPM to collect financial data out-of-the-box, however, they may also be adjusted per your business needs. The *Oracle Process Manufacturing Manufacturing Accounting Controller User's Guide* describes how to setup and use this application.

Oracle Financials Integration User's Guide

Since OPM is closely integrated with Oracle General Ledger, financial data that is collected about the manufacturing processes must be transferred to the Oracle Financials applications. The OPM Oracle Financials Integration application is where users define how that data is transferred. For example, users define whether data is transferred real time or batched and transferred at intervals. The *Oracle Process Manufacturing Oracle Financials Integration User's Guide* describes how to setup and use this application.

Inventory Management User's Guide

The OPM Inventory Management application is where data about the items purchased for, consumed during, and created as a result of the manufacturing process are tracked. The *Oracle Process Manufacturing Inventory Management User's Guide* includes information to help you effectively work with the Oracle Process Manufacturing Inventory application.

Physical Inventory User's Guide

Performing physical inventory count is the most accurate way to get an accounting of all material quantities purchased, manufactured, and sold, and update your onhand quantities accordingly. The OPM Physical Inventory application automates and enables the physical inventory process. The *Oracle Process Manufacturing Physical Inventory User's Guide* describes how to setup and use this application.

Order Fulfillment User's Guide

The OPM Order Fulfillment application automates sales order entry to reduce order cycle time. Order Fulfillment enables order entry personnel to inform customers of scheduled delivery dates and pricing. The *Oracle Process Manufacturing Order Fulfillment User's Guide* describes how to setup and use this application.

Purchase Management User's Guide

OPM Purchase Management and Oracle Purchasing combine to provide an integrated solution for Process Manufacturing. Purchase orders are entered in Oracle Purchasing and received in OPM. Then, the receipts entered in OPM are sent to Oracle Purchasing. The *Oracle Process Manufacturing Purchase Management User's Guide* describes how to setup and use this integrated solution.

Using Oracle Order Management with Process Inventory Guide

Oracle Process Manufacturing and Oracle Order Management combine to provide an integrated solution for process manufacturers. The manufacturing process is tracked and handled within Oracle Process Manufacturing, while sales orders are taken and tracked in Oracle Order Management. Process attributes, such as dual UOM and lot control, are enabled depending on the inventory organization for the item on the sales order. Order Management accepts orders entered through Oracle Customer Relationship Management (CRM). Within CRM, orders can originate from TeleSales, Sales Online, and iStore, and are booked in Order Management, making the CRM suite of products available to Process customers, through Order Management. The *Oracle Order Management User's Guide* and *Using Oracle Order Management with Process Inventory Guide* describes how to setup and use this integrated solution.

Process Execution User's Guide

The OPM Process Execution application lets you track firm planned orders and production batches from incoming materials through finished goods. Seamlessly integrated to the New Product Development application, Process Execution lets you convert firm planned orders to single or multiple production batches, allocate ingredients, record actual ingredient usage, and then complete and close production batches. Production inquiries and preformatted reports help you optimize inventory costs while maintaining a high level of customer satisfaction with on-time delivery of high quality products. The *OPM Process Execution User's Guide* presents overviews of the tasks and responsibilities for the Production Supervisor and the Production Operator. It provides prerequisite setup in other applications, and

details the windows, features, and functionality of the OPM Process Execution application.

Integration with Advanced Planning and Scheduling User's Guide

Oracle Process Manufacturing and Oracle Advanced Planning and Scheduling (APS) combine to provide an integrated solution for process manufacturers that can help increase planning efficiency. The integration provides for constraint-based planning, performance management, materials management by exception, mixed mode manufacturing that enables you to choose the best method to produce each of your products, and combine all of these methods within the same plant/company. The *Oracle Process Manufacturing Integration with Advanced Planning and Scheduling User's Guide* describes how to setup and use this application.

MPS/MRP and Forecasting User's Guide

The Oracle Process Manufacturing Material Requirements Planning (MRP) application provides long-term "views" of material demands and projected supply actions to satisfy those demands. The Master Production Scheduling (MPS) application lets you shorten that view to a much narrower and immediate time horizon, and see the immediate effects of demand and supply actions. The *Oracle Process Manufacturing MPS/MRP and Forecasting User's Guide* describes how to setup and use this application.

Capacity Planning User's Guide

The OPM Capacity Planning User's Guide describes the setup required to use OPM with the Oracle Applications Advanced Supply Chain Planning solutions. In addition, Resource setup, used by the OPM Production Execution and New Product Development applications, is also described.

Product Development User's Guide

The Oracle Process Manufacturing Product Development application provides features to manage formula and laboratory work within the process manufacturing operation. It lets you manage multiple laboratory organizations and support varying product lines throughout the organization. You can characterize and simulate the technical properties of ingredients and their effects on formulas. You can optimize formulations before beginning expensive laboratory test batches. Product Development coordinates each development function and enables a rapid, enterprise-wide implementation of new products in your plants. The *Oracle Process Manufacturing Product Development User's Guide* describes how to setup and use this application.

Quality Management User's Guide

The Oracle Process Manufacturing Quality Management application helps track the quality of ingredients and products throughout the process manufacturing cycle. Assays, also known as tests or ingredient attributes, and specifications are defined as requirements for acceptable item quality. Samples can be taken against which results are recorded for the required assays. Results can be reported through a Certificate of Analysis. The *Oracle Process Manufacturing Quality Management User's Guide* describes how to setup and use this application.

Regulatory Management User's Guide

The Oracle Process Manufacturing Regulatory Management application generates the Material Safety Data Sheets (MSDSs) required by authorities to accompany hazardous materials during shipping. You can create MSDSs from OPM Formula Management with Regulatory or Production effectivities. The *Oracle Process Manufacturing Regulatory Management User's Guide* describes how to setup and use this application.

Implementation Guide

The *Oracle Process Manufacturing Implementation Guide* offers information on setup. That is, those tasks you must complete following the initial installation of the Oracle Process Manufacturing software. Any tasks that must be completed in order to use the system out-of-the-box are included in this manual.

System Administration User's Guide

Much of the System Administration duties are performed at the Oracle Applications level, and are therefore described in the *Oracle Applications System Administrator's Guide*. The *Oracle Process Manufacturing System Administration User's Guide* provides information on the few tasks that are specific to OPM. It offers information on performing OPM file purge and archive, and maintaining such things as responsibilities, units of measure, and organizations.

API User's Guides

Public Application Programming Interfaces (APIs) are available for use with different areas of the Oracle Process Manufacturing application. APIs make it possible to pass information into and out of the application, bypassing the user interface. Use of these APIs is documented in individual manuals such as the *Oracle Process Manufacturing Inventory API User's Guide*, *Oracle Process Manufacturing Production Management and Process Operations Control APIs User's Guide*, *Oracle Process Manufacturing Formula API User's Guide*, and the *Oracle Process Manufacturing*

Cost Management API User's Guide. Additional API User's Guides are periodically added as additional public APIs are made available.

Installation and System Administration

Oracle Applications Concepts

This guide provides an introduction to the concepts, features, technology stack, architecture, and terminology for Oracle Applications Release 11*i*. It provides a useful first book to read before an installation of Oracle Applications. This guide also introduces the concepts behind Applications-wide features such as Business Intelligence (BIS), languages and character sets, and Self-Service Web Applications.

Installing Oracle Applications

This guide provides instructions for managing the installation of Oracle Applications products. In Release 11*i*, much of the installation process is handled using Oracle Rapid Install, which minimizes the time to install Oracle Applications, the Oracle8 technology stack, and the Oracle8*i* Server technology stack by automating many of the required steps. This guide contains instructions for using Oracle Rapid Install and lists the tasks you need to perform to finish your installation. You should use this guide in conjunction with individual product user's guides and implementation guides.

Upgrading Oracle Applications

Refer to this guide if you are upgrading your Oracle Applications Release 10.7 or Release 11.0 products to Release 11*i*. This guide describes the upgrade process and lists database and product-specific upgrade tasks. You must be either at Release 10.7 (NCA, SmartClient, or character mode) or Release 11.0, to upgrade to Release 11*i*. You cannot upgrade to Release 11*i* directly from releases prior to 10.7.

Maintaining Oracle Applications

Use this guide to help you run the various AD utilities, such as AutoUpgrade, AutoPatch, AD Administration, AD Controller, AD Relink, License Manager, and others. It contains how-to steps, screenshots, and other information that you need to run the AD utilities. This guide also provides information on maintaining the Oracle applications file system and database.

Oracle Applications System Administrator's Guide

This guide provides planning and reference information for the Oracle Applications System Administrator. It contains information on how to define security, customize menus and online help, and manage concurrent processing.

Oracle Alert User's Guide

This guide explains how to define periodic and event alerts to monitor the status of your Oracle Applications data.

Oracle Applications Developer's Guide

This guide contains the coding standards followed by the Oracle Applications development staff. It describes the Oracle Application Object Library components needed to implement the Oracle Applications user interface described in the *Oracle Applications User Interface Standards for Forms-Based Products*. It also provides information to help you build your custom Oracle Forms Developer 6i forms so that they integrate with Oracle Applications.

Oracle Applications User Interface Standards for Forms-Based Products

This guide contains the user interface (UI) standards followed by the Oracle Applications development staff. It describes the UI for the Oracle Applications products and how to apply this UI to the design of an application built by using Oracle Forms.

Other Implementation Documentation

Oracle Applications Product Update Notes

Use this guide as a reference for upgrading an installation of Oracle Applications. It provides a history of the changes to individual Oracle Applications products between Release 11.0 and Release 11i. It includes new features, enhancements, and changes made to database objects, profile options, and seed data for this interval.

Multiple Reporting Currencies in Oracle Applications

If you use the Multiple Reporting Currencies feature to record transactions in more than one currency, use this manual before implementing *Oracle Process Manufacturing*. This manual details additional steps and setup considerations for implementing *Oracle Process Manufacturing* with this feature.

Multiple Organizations in Oracle Applications

This guide describes how to set up and use *Oracle Process Manufacturing* with Oracle Applications' Multiple Organization support feature, so you can define and support different organization structures when running a single installation of *Oracle Process Manufacturing*.

Oracle Workflow Guide

This guide explains how to define new workflow business processes as well as customize existing Oracle Applications-embedded workflow processes. You also use this guide to complete the setup steps necessary for any Oracle Applications product that includes workflow-enabled processes.

Oracle Applications Flexfields Guide

This guide provides flexfields planning, setup and reference information for the *Oracle Process Manufacturing* implementation team, as well as for users responsible for the ongoing maintenance of Oracle Applications product data. This manual also provides information on creating custom reports on flexfields data.

Oracle eTechnical Reference Manuals

Each eTechnical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for a specific Oracle Applications product. This information helps you convert data from your existing applications, integrate Oracle Applications data with non-Oracle applications, and write custom reports for Oracle Applications products. Oracle eTRM is available on Metalink

Oracle Manufacturing APIs and Open Interfaces Manual

This manual contains up-to-date information about integrating with other Oracle Manufacturing applications and with your other systems. This documentation includes API's and open interfaces found in Oracle Manufacturing.

Oracle Order Management Suite APIs and Open Interfaces Manual

This manual contains up-to-date information about integrating with other Oracle Manufacturing applications and with your other systems. This documentation includes API's and open interfaces found in Oracle Order Management Suite.

Oracle Applications Message Reference Manual

This manual describes all Oracle Applications messages. This manual is available in HTML format on the documentation CD-ROM for Release 11i.

Training and Support

Training

Oracle offers a complete set of training courses to help you and your staff master *Oracle Process Manufacturing* and reach full productivity quickly. These courses are organized into functional learning paths, so you take only those courses appropriate to your job or area of responsibility.

You have a choice of educational environments. You can attend courses offered by Oracle University at any one of our many Education Centers, you can arrange for our trainers to teach at your facility, or you can use Oracle Learning Network (OLN), Oracle University's online education utility. In addition, Oracle training professionals can tailor standard courses or develop custom courses to meet your needs. For example, you may want to use your organization structure, terminology, and data as examples in a customized training session delivered at your own facility.

Support

From on-site support to central support, our team of experienced professionals provides the help and information you need to keep *Oracle Process Manufacturing* working for you. This team includes your Technical Representative, Account Manager, and Oracle's large staff of consultants and support specialists with expertise in your business area, managing an Oracle8i server, and your hardware and software environment.

Do Not Use Database Tools to Modify Oracle Applications Data

*Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications data unless otherwise instructed.*

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle Applications data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle Applications tables are interrelated, any change you make using Oracle Applications can update many tables at once. But when you modify Oracle Applications data using anything other than Oracle Applications, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

About Oracle

Oracle Corporation develops and markets an integrated line of software products for database management, applications development, decision support, and office automation, as well as Oracle Applications, an integrated suite of more than 160 software modules for financial management, supply chain management, manufacturing, project systems, human resources and customer relationship management.

Oracle products are available for mainframes, minicomputers, personal computers, network computers and personal digital assistants, allowing organizations to integrate different computers, different operating systems, different networks, and even different database management systems, into a single, unified computing and information resource.

Oracle is the world's leading supplier of software for information management, and the world's second largest software company. Oracle offers its database, tools, and applications products, along with related consulting, education, and support services, in over 145 countries around the world.

Your Feedback

Thank you for using *Oracle Process Manufacturing* and this user's guide.

Oracle values your comments and feedback. At the end of this guide is a Reader's Comment Form you can use to explain what you like or dislike about *Oracle Process Manufacturing* or this user's guide. Mail your comments to the following address or call us directly at (650) 506-7000.

Oracle Applications Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Or, send electronic mail to **appsdoc_us@oracle.com**.

OPM Inventory APIs

This document describes the Application Program Interfaces (APIs) that support external interfaces to the OPM Inventory tables. The topics discussed are:

- Introducing the OPM Inventory APIs
- Basic Business Needs
- Major Features
- OPM Inventory API Bill of Materials

Introducing the OPM Inventory APIs

OPM is a key component to Enterprise Resource Planning (ERP) solutions. It is a key component to Consumer Packaged Goods (CPG) and Energy and Pharmaceutical industries. In addition, OPM is deployed within customer implementations integrated to legacy systems and other solution component products.

Within this environment, the need to converge, integrate, or interface to third party systems together with other Oracle Application suites is becoming more of a business requirement, and a needed development strategy.

With the use of application program interfaces, OPM integrates with touch points either as or within an ERP solution. APIs are a documented, supported method for communicating within or between applications. The APIs support external interfaces to OPM.

These APIs support the OPM Inventory application. These interfaces make use of the standard functionality and logic implemented in that application. The APIs provide hooks where customers can interface their own programs. This guide describes how the stored procedures are called, the parameters that are required and those that are optional, and the values that are returned to the calling program. This includes error conditions that can arise.

The APIs and related functions discussed in this document are:

- Item Create API
- Item Lot/Sublot Conversion API
- Inventory Quantities API
- Lot Create API

What Is In This Document

This document describes the basic business needs, major features, architecture, and components for the insert, update, and delete features for the OPM Inventory APIs. Much of the application is divided into application-specific objects that let you link OPM functionality into your own programs. The interfaces can make use of the standard functionality and logic implemented in the OPM Inventory application.

OPM Inventory APIs are currently written in PL/SQL that are called by your own programs. To make use of these APIs, code your wrapper function that passes the appropriate parameters to the APIs. Your program is responsible for connecting to a database before calling an API function, and disconnecting from the database upon return. You can also write log files before calling and after returning from a

function. If there is a problem during execution of a call, then the APIs return one of the following status codes:

- S for success
- E for error
- U unknown or unexpected status

OPM Inventory API Features

- Creating Updating and Deleting Information
- Proper Encapsulation
- Synchronous Processing Following the Business Hierarchy
- Detailed and Translatable Error Messages

OPM Inventory API Support Policy

OPM Inventory APIs are supported by Oracle. This means:

- Oracle provides objects and libraries needed to use the APIs and the documentation for their use.
- Oracle ensures that the APIs function as designed.
- Oracle does not support customer generated programs that use the APIs.

Technical Requirements

OPM Inventory APIs are designed to operate in an OPM 11i environment only.

The procedure makes use of the following standard Oracle Applications packages:

- FND_API - the standard Oracle Applications API version checking function. This is used by the stored procedure to check for a valid API version number and also contains constant variables such as TRUE and FALSE.
- FND_MESSAGE - the standard Oracle Applications messaging function. This is used by the stored procedure to report status and error handling.
- FND_PUB_MSG - the standard Oracle Applications message retrieval function, used to search the procedure messages.

These packages are part of the 11i Oracle Applications installation. Refer to the *Oracle Applications Coding Standards* guide for further details.

Input Data Sources

Flat File

Input data to the user wrapper function comes from a flat file source. This is processed by the wrapper and header information, passed as parameters, to the stored procedure in a synchronous mode. However, along with the standard parameters, the header information is passed as a PL/SQL table. In this mode, the calling function monitors the success or failure (return code) from the called procedure. It also provides an option to COMMIT work done by the procedure.

Batch File

Input data to the user wrapper function comes from a batch file. This is processed by the wrapper and header information passed, as parameters, to the stored procedure in an asynchronous mode. In this mode, the calling function does not monitor the success or failure of each individual record. The Oracle Message FND_PUB_MSG functionality is used to audit the calls.

Online User Interface (UI)

Input data to the user wrapper function comes from a window or other user interface. This is processed by the UI and the details passed, as parameters, to the stored procedure in a synchronous mode. In this mode, the UI calling function monitors the success or failure (return code) from the called procedure. It also provides an option to COMMIT work done by the procedure.

Wrapper Function

Windows are generally used as wrapper functions. The wrapper function is responsible for collating the details required as input parameters to the stored procedure, forwarding these in the call, and monitoring the return code. The stored procedure returns three possible return code:

- S for success
- E for error
- U for unknown or unexpected status

Based on the return, the wrapper function searches the Oracle Messages File for the stored procedure to determine a COMMIT of the transaction or not.

Stored Procedure

The stored procedure is called with the appropriate parameters forwarded in a PL/SQL table format. The procedure validates each record from this table and then processes the appropriate functional logic as required. The procedure writes appropriate messages to the Oracle Messages table. These are informational as determined by the logic. These are interrogated by the calling wrapper function through the GET MESSAGES functionality.

The stored procedure calls other validation procedures in the course of its execution; a modular approach has been adopted. Functions called by these procedures do not use IN/OUT parameters. These have been removed from the Oracle 8 coding standards.

On successful completion of the procedure, a success unit is in place that can be optionally COMMITTED. The decision as to whether a COMMIT is issued on successful completion is under the control of the calling code and deliberately outside the scope of the API procedures.

Basic Business Needs

Following are some of the important characteristics that these APIs have:

Code Reuse

You can maximize code reuse from all application development tools, including PL/SQL, Oracle Forms, and Oracle Reports.

Ease of Integration

You can integrate APIs into other applications and enabling technology, such as Oracle Workflow Server, Oracle Internet Commerce & Oracle WebSystem, and Oracle EDI Gateway.

Insulation from Changes

You can encapsulate the structure of schema to prevent changing schema structures from affecting other applications.

Consistent Behavior

You can hide Object logic specific to an application from other applications, and to ensure that this logic is correctly invoked by other applications and customers.

Robust Validation

You can fully validate all incoming information into Oracle Applications.

Major Features

In order to support requirements mentioned in the “Basic Business Needs” topic, new APIs support the following functionality as business object - Item Create, Item Lot/Sublot Conversion, Inventory Quantities, and Lot Create:

Item Create API

This stored procedure is concerned with creating an inventory item function within the OPM Inventory application. The procedure is intended as a create function only, used primarily to load item data from legacy systems on implementation. The API does not let Item Update or Unit of Measure Conversion maintenance.

Item Lot/Sublot Conversion API

This stored procedure is concerned with the create an item lot/sublot unit of measure conversion function within the OPM Inventory application. The procedure is intended as a create function only. The create function is used primarily to load item data from legacy systems on implementation.

OPM Inventory Quantities API

This stored procedure is concerned with the following functions within the OPM Inventory application:

- Create inventory
- Adjust inventory
- Move inventory
- Change lot status
- Change QC grade

Lot Create API

This stored procedure is concerned with the create a lot for an inventory item function within the OPM Inventory application.

The procedure is intended as a create function only, used primarily to load item lot data from legacy systems on implementation. The Lot Create API does not do Lot Update, or Delete.

API Architecture

APIs use a layered architecture. From top to bottom these layers are:

- Wrappers and third party code - the top layer
- Public layer
- Group layer - callable from the private layer
- Private layer - where there are validation and database access layers

API Package Details

Each functional area provided by the APIs has both a public and a private package. Each possesses a package specification and a package body.

The APIs use a layered architecture. Third party and Oracle-supplied wrapper code only access the functions, procedures, and variables that are contained in the public and group package specifications. Private packages are accessed from the group code.

API packages are named according to the function they provide. The names begin with GMI followed by:

- P (public)
- V (private)
- G (group)

The next three letters define the package use, and result in the following package names:

- GMIPAPI - for the top level public API routines. These routines take in raw data in the form of PL/SQL records and make all the required database calls to find surrogate keys.
- GMIGAPI - for the group level API routines. These are called by the routines in the GMIPAPI package and are callable by third party code, if the required data is available.
- GMIGUTL - for utility routines called by all layers of the API code. They are also callable from third party and wrapper programs.

These three packages are the only ones that are called from application code. In addition, the following packages exist to support them:

- GMIVITM - for all internal processing (for example, validation) to support item creation.
- GMIVLOT - for all internal processing to support lot creation.
- GMIVILC - for all internal processing for item/lot conversion.
- GMIVQTY - for all internal processing to support item quantity processing.
- GMIVXFR - for all internal processing to support transfer processing.
- GMIVDBL - for all simple database layer processing.

Several packages are available for the more complex database processing involving inventory transactions for the `ic_summ_inv`, `ic_loct_inv`, `ic_tran_pnd` and `ic_tran_cmp` tables as follows:

- GMIPTXN - Public transaction engine API routines. These are the only routines in the transaction processor that are called publicly.
- GMIVTXN - Private transaction engine processing routines.
- GMIVBUS - Business logic for `ic_summ_inv` manipulation.
- GMIVBUL - Business logic for `ic_loct_inv` manipulation.
- GMIVPND - Database routines for `ic_tran_pnd`.
- GMIVCMP - Database routines for `ic_tran_cmp`.
- GMIVSUM - Database routines for `ic_summ_inv`.
- GMIVLOC - Database routines for `ic_loct_inv`.

The API release also includes some wrappers that are used to process flat data files. They are included more as an example of how the public APIs are called rather than a supported product, since they have to be modified to suit a particular installation.

The wrappers are:

- GMIPITW - Wrapper for item creation.
- GMIPLOW - Wrapper for lot creation.
- GMIPILW - Wrapper for item/lot UOM conversion creation.
- GMIPQTW - Wrapper for quantity transactions.

When the intracompany transfer package is finalized, there is a wrapper (GMIPXFW).

The files that contain the packages take the previously stated names and add an S for the specification or B for the package body. The file type is .pls and it is written in lowercase letters.

OPM Inventory API Bill of Materials

The following is a list of packages and files that are delivered with the OPM Inventory APIs. These must be on your system for your interface to compile and link properly.

Package Name	File Names	Description
GMF_ItemCost_PUB	GMFPCSTS.pls GMFPCSTB.pls	Public Cost Management package that the user defined function calls. The business API is used for inserting, updating, deleting, or retrieving an item cost.
GMF_ResourceCost_PUB	GMFPRESS.pls GMFPCSTB.pls	Public Cost Management package that the user defined function calls. The business API is used for inserting, updating, deleting, or retrieving a resource cost.
GMF_AllocationDefinition_PUB	GMFPALCS.pls GMFPCSTB.pls	Public Cost Management package that the user defined function calls. The business API is used for inserting, updating, or deleting an allocation definition.
GMF_BurdenDetails_PUB	GMFPBRDS.pls GMFPBRDB.pls	Public Cost Management package that the user defined function calls. The business API is used for inserting, updating, deleting, or retrieving a burden detail.

OPM Inventory API Usage

The OPM Inventory APIs are written in PL/SQL. To make use of these APIs, code your interface or wrapper. Your program is responsible for connecting to a database before calling an API function. You can also write to log files before calling and after returning from an API function. Each function returns an error code in the parameter `x_return_status` that indicates whether the API was successful or failed. The values are as follows:

- S for success
- E for error
- U unknown or unexpected status

The topics discussed are:

- Item Create API Wrapper - Business Function
- Calling the API Interface Code
- Item Create API Wrapper - Input Structure
- Item Create API Wrapper - ASCII Flat File Layout
- Log Files
- Calling the API Code - Example
- Item Create API Wrapper - Code Example

Item Create API Wrapper - Business Function

This stored procedure is designed to operate in conjunction with the item create API that is used to create an item in OPM. It can be required for use in both synchronous (online) and asynchronous (batch) modes. When used in synchronous mode, the calling program (for example, an Oracle window) calls the API directly.

This topic discusses using the API in asynchronous mode through a wrapper function. The source of data for the wrapper comes from an ASCII flat file.

This topic describes how the wrapper function is called and the parameters that are required and optional.

Wrapper Function - Input Data Sources

The following are input data sources for the wrapper function.

Flat File

Input data to the user wrapper function can come from a flat file source. The flat file is read record by record. For each record, the wrapper builds the necessary parameters and calls the API. The wrapper assumes no interdependency between the records and therefore instructs the API to commit each successfully processed record.

Temporary Table

Input data to the user wrapper function can come from a temporary database table. The table is read row by row. For each row the wrapper builds the necessary parameters and calls the API. The wrapper assumes no interdependency between the rows and therefore instructs the API to commit each successfully processed row.

Stored Procedures Overview

Stored procedures are held at the database level and consist of PL/SQL-based routines and functions. These are therefore application independent and are called from any privileged program function accessing the database. At the functional level, stored procedures manage both the business rules and the data.

A stored procedure has two parts:

- Specification - declares the procedure or function name with parameters.
- Body - defines the procedure to execute consisting of PL/SQL block statements.

Stored procedures offer a number of benefits:

Compiled Version of PL/SQL in the Database

Stored procedures are compiled and held at the database level. Therefore, they occur once only and can be called by all privileged users. The code is compiled the first time the stored procedure is called.

When called, there is no parsing of the statement to interpret the PL/SQL. All dependencies and syntax of the procedure are already defined aiding in efficiency and performance.

Security Considerations

Grant permissions can be controlled at the stored procedure level, above the individual database tables. This lets a global level of authorization against the procedure rather than a table by table grant discipline within an application that holds anomalies. If you have grant authority to the procedure, then it is assumed you have permissions at the database and application levels.

Platform Independence

Stored procedure, unlike C coded functions, are platform independent with no need to recompile according to hardware and operating platform.

Reduced Network Traffic

Stored procedures reduce network traffic since only the execute statement and parameter values need to be passed over the network.

Separation from User Interface (UI) Layer

Stored procedures are defined at the database level, and are away from the UI layer of the application software.

Stored Procedure Execution

The stored procedure can be called with the appropriate parameters forwarded in a PL/SQL RECORD format. The procedure validates the RECORD and then processes the appropriate functional logic as required. The procedure writes appropriate messages to the Oracle Messages table. These are informational (succeeded) or error as determined by the logic. These can be interrogated by the calling wrapper function using the GET MESSAGES functionality.

The stored procedure can call other validation procedures in the course of its execution (for example, validate UOM code).

On successful completion of the procedure, the COMMIT of the database updates is made. The procedure is viewed as one LOGICAL transaction. The COMMIT is used if the procedure executes successfully.

Calling the API Interface Code

The following are part of a sample wrapper and are used to test the API code. Wrappers are written in PL/SQL Package. Wrappers can be written for each API and call the APIs. The source of data for the wrapper comes from an ASCII flat file. You can write a similar type of wrapper to call the API code.

These wrappers have the following parameters:

p_dir IN VARCHAR2 - Working directory for input and output files.

p_input_file IN VARCHAR2 - Name of input file (for example, ASCII flat file with data).

p_output_file IN VARCHAR2 - Name of output file.

p_delimiter IN VARCHAR2 - Delimiter character.

The ASCII flat file must be character delimited, typically with a comma.

Item Create API Wrapper - Input Structure

The API wrapper consists of a PL/SQL procedure and PL/SQL function both named `Create_Item`.

Item Create API Wrapper - ASCII Flat File Layout

The ASCII flat file must be character delimited, typically with a comma.

Omitting an optional field is achieved by leaving the column positions as spaces (for fixed format files) or using consecutive delimiters (for delimited files).

Log Files

When an API is run, it generates a log file that describes what was generated during the API call or what went wrong. These log files must be placed in a specific directory on your system (determined by a parameter in your `init.ora` file) and that directory must have open permissions to write to that directory. To know where this directory is, run the following query.

```
SQL> select value from v$parameter where name like 'utl%';
```

This query can return multiple directories. Ensure that the directory you want the log files written to exists, and that the permissions are open to write to it. This directory is entered in your wrapper file where you also determine what the log file is called.

There is also a log file that is generated with the session number that can provide some additional information. The log file that is system generated is `wrapperxxxxx.log` where the `xxxxx` is the session number. These two log files have similar information, but if you encounter problems, then this can be another place to look for information.

Calling the API Code - Example

This section details how to call the API code within the wrapper. The purpose of this is to explain how to call a standard OPM Inventory API.

Item Create API Wrapper - Code Example

The following is an example of the PL/SQL code for the Item Create API wrapper.

```

WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
CREATE OR REPLACE PACKAGE BODY GMI_ITEM_WRP AS
-- $Header: GMIPITWB.pls 115.6 2000/09/27 19:20:16 mpetrosi gmigapib.pls $
-- Body start of comments
-----+-----
--| PROCEDURE NAME
--|   Create_Item
--|
--| TYPE
--|   Public
--|
--| USAGE
--|   Create an Inventory Item
--|
--| DESCRIPTION
--|   This is a PL/SQL wrapper procedure to call the Create_Item
--|   API wrapper function
--|
--| PARAMETERS
--|   p_dir           IN VARCHAR2           - Working directory for input
--|                                     and output files.
--|   p_input_file    IN VARCHAR2           - Name of input file
--|   p_output_file   IN VARCHAR2           - Name of output file
--|   p_delimiter     IN VARCHAR2           - Delimiter character
--|
--| RETURNS
--|   None
--|
--| HISTORY
--|   16-AUG-1999      B965832(1)  Set lot_status/qc_grade to NULL if
--|                                     they are read in as spaces
-----+-----
-- Api end of comments
PROCEDURE Create_Item
( p_dir           IN VARCHAR2
, p_input_file    IN VARCHAR2
, p_output_file   IN VARCHAR2
, p_delimiter     IN VARCHAR2
)
IS

```

```

l_return_status VARCHAR2(1);

BEGIN

l_return_status :=Create_item( p_dir
    , p_input_file
                                , p_output_file
                                , p_delimiter
                                );

End Create_Item;

```

```

---+=====+
--| FUNCTION NAME
--|   Create_Item
--|
--| TYPE
--|   Public
--|
--| USAGE
--|   Create an inventory item
--|
--| DESCRIPTION
--|   This is a PL/SQL wrapper function to call the FND
--|   Inventory Create Item API.
--|   It reads item data from a flat file and outputs any error
--|   messages to a second flat file. It also generates a Status
--|   called wrapper<session_id>.log in the /tmp directory.
--|
--| PARAMETERS
--|   p_dir           IN VARCHAR2           - Working directory for input
--|                                     and output files.
--|   p_input_file    IN VARCHAR2           - Name of input file
--|   p_output_file   IN VARCHAR2           - Name of output file
--|   p_delimiter     IN VARCHAR2           - Delimiter character
--|
--| RETURNS
--|   VARCHAR2 - 'S' All records processed successfully
--|             'E' 1 or more records errored
--|             'U' 1 or more record unexpected error
--|
--| HISTORY
--|
---+=====+

```

```
-- Api end of comments
FUNCTION Create_Item
( p_dir          IN VARCHAR2
, p_input_file   IN VARCHAR2
, p_output_file  IN VARCHAR2
, p_delimiter    IN VARCHAR2
)
RETURN VARCHAR2
IS

--
-- Local variables
--

l_status          VARCHAR2(1);
l_return_status   VARCHAR2(1) :=FND_API.G_RET_STS_SUCCESS;
l_count           NUMBER   ;
l_record_count    NUMBER   :=0;
l_loop_cnt        NUMBER   :=0;
l_dummy_cnt       NUMBER   :=0;
l_data            VARCHAR2(2000);
item_rec          GMIGAPI.item_rec_typ;
l_ic_item_mst_row ic_item_mst%ROWTYPE;
l_ic_item_cpg_row ic_item_cpg%ROWTYPE;
l_p_dir           VARCHAR2(50);
l_output_file     VARCHAR2(20);
l_outfile_handle  UTL_FILE.FILE_TYPE;
l_input_file      VARCHAR2(20);
l_infile_handle   UTL_FILE.FILE_TYPE;
l_line            VARCHAR2(800);
l_delimiter       VARCHAR(1);
l_log_dir         VARCHAR2(50);
l_log_name        VARCHAR2(20) :='wrapper';
l_log_handle      UTL_FILE.FILE_TYPE;
l_global_file     VARCHAR2(20);

l_session_id      VARCHAR2(10);

BEGIN

-- Enable The Buffer
DBMS_OUTPUT.ENABLE(1000000);

l_p_dir           :=p_dir;
l_input_file      :=p_input_file;
```

```
l_output_file      :=p_output_file;
l_delimiter        :=p_delimiter;
l_global_file      :=l_input_file;

--
-- Obtain The SessionId To Append To wrapper File Name.
--

l_session_id := USERENV('sessionid');

l_log_name := CONCAT(l_log_name,l_session_id);
l_log_name := CONCAT(l_log_name, '.log');

--
-- Directory is now the same same as for the out file
--
l_log_dir := p_dir;

--
-- Open The Wrapper File For Output And The Input File for Input.
--

l_log_handle      :=UTL_FILE.FOPEN(l_log_dir, l_log_name, 'w');
l_infile_handle   :=UTL_FILE.FOPEN(l_p_dir, l_input_file, 'r');

--
-- Loop thru flat file and call Inventory Quantities API
--

dms_output.put_line('Start Processing');
UTL_FILE.PUT_LINE(l_log_handle, 'Process Started at '
|| to_char(SYSDATE, 'DD-MON-YY HH24:MI:SS'));

UTL_FILE.NEW_LINE(l_log_handle);
UTL_FILE.PUT_LINE(l_log_handle, 'Input Directory ' || l_p_dir );
UTL_FILE.PUT_LINE(l_log_handle, 'Input File ' || l_input_file );
UTL_FILE.PUT_LINE(l_log_handle, 'Record Type ' || l_delimiter );
UTL_FILE.PUT_LINE(l_log_handle, 'Output File ' || l_output_file );

l_outfile_handle :=UTL_FILE.FOPEN(l_p_dir, l_output_file, 'w');

LOOP
l_record_count :=l_record_count+1;
```

```

BEGIN
  UTL_FILE.GET_LINE(l_infile_handle, l_line);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
EXIT;
  END;

  UTL_FILE.NEW_LINE(l_log_handle);
  UTL_FILE.PUT_LINE(l_log_handle, 'Reading Record ' || l_record_count );

  item_rec.item_no          :=Get_Field(l_line,l_delimiter,1);
  item_rec.item_desc1      :=Get_Field(l_line,l_delimiter,2);
  item_rec.item_desc2      :=Get_Field(l_line,l_delimiter,3);
  item_rec.alt_itema       :=Get_Field(l_line,l_delimiter,4);
  item_rec.alt_itemb       :=Get_Field(l_line,l_delimiter,5);
  item_rec.item_um         :=Get_Field(l_line,l_delimiter,6);
  item_rec.dualum_ind      :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,7),' '), ' ', '0'));
  item_rec.item_um2        :=Get_Field(l_line,l_delimiter,8);
  item_rec.deviation_lo    :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,9),' '), ' ', '0'));
  item_rec.deviation_hi    :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,10),' '), ' ', '0'));
  item_rec.level_code      :=TO_NUMBER(Get_Field(l_line,l_delimiter,11));
  item_rec.lot_ctl         :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,12),' '), ' ', '0'));
  item_rec.lot_indivisible :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,13),' '), ' ', '0'));
  item_rec.sublot_ctl      :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,14),' '), ' ', '0'));
  item_rec.loct_ctl        :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,15),' '), ' ', '0'));
  item_rec.noninv_ind      :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,16),' '), ' ', '0'));
  item_rec.match_type      :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,17),' '), ' ', '0'));
  item_rec.inactive_ind   :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,18),' '), ' ', '0'));
  item_rec.inv_type        :=Get_Field(l_line,l_delimiter,19);
  item_rec.shelf_life      :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,20),' '), ' ', '0'));
  item_rec.retest_interval :=
  TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,21),' '), ' ', '0'));
  item_rec.item_abccode    :=Get_Field(l_line,l_delimiter,22);

```

```
item_rec.gl_class      :=Get_Field(l_line,l_delimiter,23);
item_rec.inv_class    :=Get_Field(l_line,l_delimiter,24);
item_rec.sales_class  :=Get_Field(l_line,l_delimiter,25);
item_rec.ship_class   :=Get_Field(l_line,l_delimiter,26);
item_rec.frt_class    :=Get_Field(l_line,l_delimiter,27);
item_rec.price_class  :=Get_Field(l_line,l_delimiter,28);
item_rec.storage_class :=Get_Field(l_line,l_delimiter,29);
item_rec.purch_class  :=Get_Field(l_line,l_delimiter,30);
item_rec.tax_class    :=Get_Field(l_line,l_delimiter,31);
item_rec.customs_class :=Get_Field(l_line,l_delimiter,32);
item_rec.alloc_class  :=Get_Field(l_line,l_delimiter,33);
item_rec.planning_class :=Get_Field(l_line,l_delimiter,34);
item_rec.itemcost_class :=Get_Field(l_line,l_delimiter,35);
item_rec.cost_mthd_code :=Get_Field(l_line,l_delimiter,36);
item_rec.upc_code     :=Get_Field(l_line,l_delimiter,37);
item_rec.grade_ctl    :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,38),' '), ' ', '0'));
item_rec.status_ctl   :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,39),' '), ' ', '0'));
item_rec.qc_grade     :=Get_Field(l_line,l_delimiter,40);
--B965832(1) Check for spaces
IF item_rec.qc_grade = ' '
THEN
item_rec.qc_grade := '';
END IF;
--B965832(1) End
item_rec.lot_status   :=Get_Field(l_line,l_delimiter,41);
--B965832(1) Check for spaces
IF item_rec.lot_status = ' '
THEN
item_rec.lot_status := '';
END IF;
--B965832(1) End
item_rec.bulk_id      :=TO_NUMBER(Get_Field(l_line,l_delimiter,42));
item_rec.pkg_id       :=TO_NUMBER(Get_Field(l_line,l_delimiter,43));
item_rec.qcitem_no    :=Get_Field(l_line,l_delimiter,44);
item_rec.qchold_res_code :=Get_Field(l_line,l_delimiter,45);
item_rec.expaction_code :=Get_Field(l_line,l_delimiter,46);
item_rec.fill_qty     :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,47),' '), ' ', '0'));
item_rec.fill_um      :=Get_Field(l_line,l_delimiter,48);
item_rec.expaction_interval :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,49),' '), ' ', '0'));
item_rec.phantom_type :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,50),' '), ' ', '0'));
```

```
item_rec.whse_item_no      :=Get_Field(l_line,l_delimiter,51);
item_rec.experimental_ind:=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,52),' '),' ','0'));
IF (Get_Field(l_line,l_line,53) IS NULL)
THEN
    item_rec.exported_date :=TO_DATE('02011970','DDMMYYYY');
ELSE
    item_rec.exported_date :=TO_DATE(
    Get_Field(l_line,l_delimiter,53),'DDMMYYYY');
END IF;
item_rec.seq_dpnd_class   :=Get_Field(l_line,l_delimiter,54);
item_rec.commodity_code  :=Get_Field(l_line,l_delimiter,55);
item_rec.ic_matr_days    :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,56),' '),' ','0'));
item_rec.ic_hold_days   :=
TO_NUMBER(TRANSLATE(NVL(Get_Field(l_line,l_delimiter,57),' '),' ','0'));
IF ((Get_Field(l_line,l_delimiter,58)) IS NULL)
THEN
    item_rec.user_name    :='OPM';
ELSE
    item_rec.user_name    :=Get_Field(l_line,l_delimiter,58);
END IF;
item_rec.attribute1      :=Get_Field(l_line,l_delimiter,59);
item_rec.attribute2      :=Get_Field(l_line,l_delimiter,60);
item_rec.attribute3      :=Get_Field(l_line,l_delimiter,61);
item_rec.attribute4      :=Get_Field(l_line,l_delimiter,62);
item_rec.attribute5      :=Get_Field(l_line,l_delimiter,63);
item_rec.attribute6      :=Get_Field(l_line,l_delimiter,64);
item_rec.attribute7      :=Get_Field(l_line,l_delimiter,65);
item_rec.attribute8      :=Get_Field(l_line,l_delimiter,66);
item_rec.attribute9      :=Get_Field(l_line,l_delimiter,67);
item_rec.attribute10     :=Get_Field(l_line,l_delimiter,68);
item_rec.attribute11     :=Get_Field(l_line,l_delimiter,69);
item_rec.attribute12     :=Get_Field(l_line,l_delimiter,70);
item_rec.attribute13     :=Get_Field(l_line,l_delimiter,71);
item_rec.attribute14     :=Get_Field(l_line,l_delimiter,72);
item_rec.attribute15     :=Get_Field(l_line,l_delimiter,73);
item_rec.attribute16     :=Get_Field(l_line,l_delimiter,74);
item_rec.attribute17     :=Get_Field(l_line,l_delimiter,75);
item_rec.attribute18     :=Get_Field(l_line,l_delimiter,76);
item_rec.attribute19     :=Get_Field(l_line,l_delimiter,77);
item_rec.attribute20     :=Get_Field(l_line,l_delimiter,78);
item_rec.attribute21     :=Get_Field(l_line,l_delimiter,79);
item_rec.attribute22     :=Get_Field(l_line,l_delimiter,80);
item_rec.attribute23     :=Get_Field(l_line,l_delimiter,81);
```

```
item_rec.attribute24      :=Get_Field(l_line,l_delimiter,82);
item_rec.attribute25      :=Get_Field(l_line,l_delimiter,83);
item_rec.attribute26      :=Get_Field(l_line,l_delimiter,84);
item_rec.attribute27      :=Get_Field(l_line,l_delimiter,85);
item_rec.attribute28      :=Get_Field(l_line,l_delimiter,86);
item_rec.attribute29      :=Get_Field(l_line,l_delimiter,87);
item_rec.attribute30      :=Get_Field(l_line,l_delimiter,88);
item_rec.attribute_category :=Get_Field(l_line,l_delimiter,89);

UTL_FILE.PUT_LINE(l_log_handle,'item_no           = ' || item_rec.item_no);
UTL_FILE.PUT_LINE(l_log_handle,'item_desc1        = ' || item_rec.item_desc1);
UTL_FILE.PUT_LINE(l_log_handle,'item_desc2        = ' || item_rec.item_desc2);
UTL_FILE.PUT_LINE(l_log_handle,'alt_itema         = ' || item_rec.alt_itema);
UTL_FILE.PUT_LINE(l_log_handle,'alt_itemb         = ' || item_rec.alt_itemb);
UTL_FILE.PUT_LINE(l_log_handle,'item_um          = ' || item_rec.item_um);
UTL_FILE.PUT_LINE(l_log_handle,'dualum_ind        = ' || item_rec.dualum_ind);
UTL_FILE.PUT_LINE(l_log_handle,'item_um2         = ' || item_rec.item_um2);
UTL_FILE.PUT_LINE(l_log_handle,'deviation_lo      = ' || item_rec.deviation_lo);
UTL_FILE.PUT_LINE(l_log_handle,'deviation_hi      = ' || item_rec.deviation_hi);
UTL_FILE.PUT_LINE(l_log_handle,'level_code       = ' || item_rec.level_code);
UTL_FILE.PUT_LINE(l_log_handle,'lot_ctl          = ' || item_rec.lot_ctl);
UTL_FILE.PUT_LINE(l_log_handle,'lot_indivisible= ' || item_rec.lot_indivisible);
UTL_FILE.PUT_LINE(l_log_handle,'sublot_ctl        = ' || item_rec.sublot_ctl);
UTL_FILE.PUT_LINE(l_log_handle,'loct_ctl         = ' || item_rec.loct_ctl);
UTL_FILE.PUT_LINE(l_log_handle,'noninv_ind       = ' || item_rec.noninv_ind);
UTL_FILE.PUT_LINE(l_log_handle,'match_type       = ' || item_rec.match_type);
UTL_FILE.PUT_LINE(l_log_handle,'inactive_ind     = ' || item_rec.inactive_ind);
UTL_FILE.PUT_LINE(l_log_handle,'inv_type         = ' || item_rec.inv_type);
UTL_FILE.PUT_LINE(l_log_handle,'shelf_life       = ' || item_rec.shelf_life);
UTL_FILE.PUT_LINE(l_log_handle,'retest_interval= ' || item_rec.retest_interval);
UTL_FILE.PUT_LINE(l_log_handle,'item_abccode     = ' || item_rec.item_abccode);
UTL_FILE.PUT_LINE(l_log_handle,'gl_class        = ' || item_rec.gl_class);
UTL_FILE.PUT_LINE(l_log_handle,'inv_class       = ' || item_rec.inv_class);
UTL_FILE.PUT_LINE(l_log_handle,'sales_class     = ' || item_rec.sales_class);
UTL_FILE.PUT_LINE(l_log_handle,'ship_class      = ' || item_rec.ship_class);
UTL_FILE.PUT_LINE(l_log_handle,'fprt_class     = ' || item_rec.fprt_class);
UTL_FILE.PUT_LINE(l_log_handle,'price_class    = ' || item_rec.price_class);
UTL_FILE.PUT_LINE(l_log_handle,'storage_class  = ' || item_rec.storage_class);
UTL_FILE.PUT_LINE(l_log_handle,'purch_class   = ' || item_rec.purch_class);
UTL_FILE.PUT_LINE(l_log_handle,'tax_class     = ' || item_rec.tax_class);
UTL_FILE.PUT_LINE(l_log_handle,'customs_class = ' || item_rec.customs_class);
UTL_FILE.PUT_LINE(l_log_handle,'alloc_class   = ' || item_rec.alloc_class);
UTL_FILE.PUT_LINE(l_log_handle,'planning_class = ' || item_rec.planning_class);
UTL_FILE.PUT_LINE(l_log_handle,'itemcost_class = ' || item_rec.itemcost_class);
UTL_FILE.PUT_LINE(l_log_handle,'cost_mthd_code = ' || item_rec.cost_mthd_code);
```

```
UTL_FILE.PUT_LINE(l_log_handle, 'upc_code           = ' || item_rec.upc_code);
UTL_FILE.PUT_LINE(l_log_handle, 'grade_ctl        = ' || item_rec.grade_ctl);
UTL_FILE.PUT_LINE(l_log_handle, 'status_ctl       = ' || item_rec.status_ctl);
UTL_FILE.PUT_LINE(l_log_handle, 'qc_grade        = ' || item_rec.qc_grade);
UTL_FILE.PUT_LINE(l_log_handle, 'lot_status      = ' || item_rec.lot_status);
UTL_FILE.PUT_LINE(l_log_handle, 'bulk_id        = ' || item_rec.bulk_id);
UTL_FILE.PUT_LINE(l_log_handle, 'pkg_id         = ' || item_rec.pkg_id);
UTL_FILE.PUT_LINE(l_log_handle, 'qcitem_no      = ' || item_rec.qcitem_no);
UTL_FILE.PUT_LINE(l_log_handle, 'qchold_res_code= ' || item_rec.qchold_res_code);
UTL_FILE.PUT_LINE(l_log_handle, 'expaction_code = ' || item_rec.expaction_code);
UTL_FILE.PUT_LINE(l_log_handle, 'fill_qty       = ' || item_rec.fill_qty);
UTL_FILE.PUT_LINE(l_log_handle, 'fill_um       = ' || item_rec.fill_um);
UTL_FILE.PUT_LINE(
    l_log_handle, 'expaction_interval = ' || item_rec.expaction_interval);
UTL_FILE.PUT_LINE(l_log_handle, 'phantom_type    = ' || item_rec.phantom_type);
UTL_FILE.PUT_LINE(l_log_handle, 'whse_item_no   = ' || item_rec.whse_item_no);
UTL_FILE.PUT_LINE(
    l_log_handle, 'experimental_ind = ' || item_rec.experimental_ind);
UTL_FILE.PUT_LINE(l_log_handle, 'exported_date  = ' || item_rec.exported_date);
UTL_FILE.PUT_LINE(l_log_handle, 'seq_dpnd_class = ' || item_rec.seq_dpnd_class);
UTL_FILE.PUT_LINE(l_log_handle, 'commodity_code = ' || item_rec.commodity_code);
UTL_FILE.PUT_LINE(l_log_handle, 'ic_matr_days  = ' || item_rec.ic_matr_days);
UTL_FILE.PUT_LINE(l_log_handle, 'ic_hold_days  = ' || item_rec.ic_hold_days);
UTL_FILE.PUT_LINE(l_log_handle, 'user_name    = ' || item_rec.user_name);
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute1   = ' ||
    item_rec.attribute1 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute2   = ' ||
    item_rec.attribute2 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute3   = ' ||
    item_rec.attribute3 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute4   = ' ||
    item_rec.attribute4 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute5   = ' ||
    item_rec.attribute5 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute6   = ' ||
    item_rec.attribute6 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute7   = ' ||
    item_rec.attribute7 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute8   = ' ||
    item_rec.attribute8 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute9   = ' ||
    item_rec.attribute9 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute10  = ' ||
    item_rec.attribute10 );
UTL_FILE.PUT_LINE(l_log_handle, 'Attribute11  = ' ||
```

```
item_rec.attribute11 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute12 = ' ||
item_rec.attribute12 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute13 = ' ||
item_rec.attribute13 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute14 = ' ||
item_rec.attribute14 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute15 = ' ||
item_rec.attribute15 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute16 = ' ||
item_rec.attribute16 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute17 = ' ||
item_rec.attribute17 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute18 = ' ||
item_rec.attribute18 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute19 = ' ||
item_rec.attribute19 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute20 = ' ||
item_rec.attribute20 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute21 = ' ||
item_rec.attribute21 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute22 = ' ||
item_rec.attribute22 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute23 = ' ||
item_rec.attribute23 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute24 = ' ||
item_rec.attribute24 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute25 = ' ||
item_rec.attribute25 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute26 = ' ||
item_rec.attribute26 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute27 = ' ||
item_rec.attribute27 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute28 = ' ||
item_rec.attribute28 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute29 = ' ||
item_rec.attribute29 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute30 = ' ||
item_rec.attribute30 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute_Category = ' ||
item_rec.attribute_category );

GMIPAPI.Create_Item
( p_api_version      => 3.0
, p_init_msg_list    => FND_API.G_TRUE
```

```
, p_commit          => FND_API.G_TRUE
, p_validation_level => FND_API.G_VALID_LEVEL_FULL
, p_item_rec        => item_rec
, x_ic_item_mst_row => l_ic_item_mst_row
, x_ic_item_cpg_row => l_ic_item_cpg_row
, x_return_status   => l_status
, x_msg_count       => l_count
, x_msg_data        => l_data
);

IF l_count > 0
THEN
  l_loop_cnt :=1;
  LOOP

    FND_MSG_PUB.Get(
      p_msg_index    => l_loop_cnt,
      p_data         => l_data,
      p_encoded      => FND_API.G_FALSE,
      p_msg_index_out => l_dummy_cnt);

    -- dbms_output.put_line('Message ' || l_data );

    UTL_FILE.PUT_LINE(l_outfile_handle, 'Record = ' || l_record_count );
    UTL_FILE.PUT_LINE(l_outfile_handle, l_data);
    UTL_FILE.NEW_LINE(l_outfile_handle);

    IF l_status = 'E' OR
       l_status = 'U'
    THEN
      l_data := CONCAT('ERROR ', l_data);
    END IF;

    UTL_FILE.PUT_LINE(l_log_handle, l_data);

    -- Update error status
    IF (l_status = 'U')
    THEN
      l_return_status := l_status;
    ELSIF (l_status = 'E' and l_return_status <> 'U')
    THEN
      l_return_status := l_status;
    ELSE
      l_return_status := l_status;
    END IF;
```

```
l_loop_cnt := l_loop_cnt + 1;
IF l_loop_cnt > l_count
THEN
    EXIT;
END IF;

END LOOP;

END IF;

END LOOP;
    UTL_FILE.NEW_LINE(l_log_handle);
    UTL_FILE.PUT_LINE(l_log_handle, 'Process Completed at '
    || to_char(SYSDATE, 'DD-MON-YY HH24:MI:SS'));
--
-- Check if any messages generated. If so then decode and
-- output to error message flat file
--

UTL_FILE.FCLOSE_ALL;

RETURN l_return_status;

EXCEPTION
WHEN UTL_FILE.INVALID_OPERATION THEN
    dbms_output.put_line('Invalid Operation For ' || l_global_file);
    UTL_FILE.FCLOSE_ALL;
    RETURN l_return_status;

WHEN UTL_FILE.INVALID_PATH THEN
    dbms_output.put_line('Invalid Path For ' || l_global_file);
    UTL_FILE.FCLOSE_ALL;
    RETURN l_return_status;

WHEN UTL_FILE.INVALID_MODE THEN
    dbms_output.put_line('Invalid Mode For ' || l_global_file);
    UTL_FILE.FCLOSE_ALL;
    RETURN l_return_status;

WHEN UTL_FILE.INVALID_FILEHANDLE THEN
    dbms_output.put_line('Invalid File Handle ' || l_global_file);
    UTL_FILE.FCLOSE_ALL;
    RETURN l_return_status;
```



```
--| HISTORY |
--| |
--+-----+
-- Api end of comments
FUNCTION Get_Field
( p_line      IN VARCHAR2
, p_delimiter IN VARCHAR2
, p_field_no  IN NUMBER
)
RETURN VARCHAR2
IS
--
-- Local variables
--
l_start      NUMBER :=0;
l_end        NUMBER :=0;

BEGIN

-- Determine start position
IF p_field_no = 1
THEN
    l_start      :=0;
ELSE
    l_start      :=INSTR(p_line,p_delimiter,1,(p_field_no - 1));
    IF l_start    = 0
    THEN
        RETURN NULL;
    END IF;
END IF;

-- Determine end position
l_end         :=INSTR(p_line,p_delimiter,1,p_field_no);
IF l_end      = 0
THEN
    l_end         := LENGTH(p_line) + 1;
END IF;

-- Extract the field data
IF (l_end - l_start) = 1
THEN
    RETURN NULL;
ELSE
    RETURN SUBSTR(p_line,(l_start + 1),((l_end - l_start) - 1));
END IF;
```

```

EXCEPTION
  WHEN OTHERS
  THEN
    RETURN NULL;

END Get_Field;

--+-----+
--| FUNCTION NAME
--|   Get_Substring
--|
--| TYPE
--|   Public
--|
--| USAGE
--|   Get value of Sub-string from formatted ASCII data file record
--|
--| DESCRIPTION
--|   This utility function will return the value of a passed sub-string
--|   of a formatted ASCII data file record
--|
--| PARAMETERS
--|   p_substring          IN VARCHAR2          - substring data
--|
--| RETURNS
--|   VARCHAR2            - Value of field
--|
--| HISTORY
--|
--+-----+
-- Api end of comments
FUNCTION Get_Substring
( p_substring    IN VARCHAR2
)
RETURN VARCHAR2
IS
--
-- Local variables
--
l_string_value  VARCHAR2(200) := ' ';

BEGIN

-- Determine start position

```

```
l_string_value :=NVL(RTRIM(LTRIM(p_substring)), ' ');

RETURN l_string_value;
EXCEPTION
  WHEN OTHERS
  THEN
    RETURN ' ';

END Get_Substring;

END GMI_ITEM_WRP;
/
-- show errors;
COMMIT;
EXIT;
```

Technical Overview

This section describes the technical information associated to an API, such as the structure and files included.

The topics discussed are:

- Item Create
- Item Lot/Sublot Conversion
- Inventory Quantities
- Lot Create
- Standard Parameters

Item Create

The Item Create stored procedure is intended to be used by a user wrapper calling function with item attributes passed to the procedure using a RECORD format. The wrapper function is responsible for connecting to the database as an appropriate user with the necessary privileges. It passes the appropriate parameters into the stored procedure and is responsible for handling the return code from the procedure.

According to API standards, the following are the names of files, packages, and procedures for Public APIs:

Structure for Item Create Public APIs

Object Type	Name
Package Specification File	GMFPCSTS.pls
Package Body File	GMFPCSTB.pls
Package	GMI_ItemCreate_PUB
Procedure - Item Create	Create_Item

Item Lot/Sublot Conversion

The Item Lot Conversion stored procedure is intended to be used by a user wrapper calling function with item attributes passed to the procedure through a RECORD format. The wrapper function is responsible for connecting to the database as an appropriate user with the necessary privileges. It passes the appropriate parameters into the stored procedure and is responsible for handling the return code from the procedure.

Structure for Item Lot/Sublot Conversion Public APIs

Object Type	Name
Package Specification File	GMFPCSTS.pls
Package Body File	GMFPCSTB.pls
Package	GMI_ItemCreate_PUB
Procedure - Item Create	Create_Item

Inventory Quantities

The Inventory Quantities stored procedure is intended to be used by a user wrapper calling function with item attributes passed to the procedure through a RECORD format. The wrapper function is responsible for connecting to the database as an appropriate user with the necessary privileges. It passes the appropriate parameters into the stored procedure and is responsible for handling the return code from the procedure.

Structure for Inventory Quantities Public APIs

Object Type	Name
Package Specification File	GMFPCSTS.pls
Package Body File	GMFPCSTB.pls
Package	GMI_ItemCreate_PUB
Procedure - Item Create	Create_Item

Lot Create

The Lot Create stored procedure is intended to be used by a user wrapper calling function with lot attributes passed to the procedure using a RECORD format. The wrapper function passes the appropriate parameters into the stored procedure and is responsible for handling the return code from the procedure.

Structure for Lot Create Public APIs

Object Type	Name
Package Specification File	GMFPCSTS.pls
Package Body File	GMFPCSTB.pls
Package	GMI_ItemCreate_PUB
Procedure - Item Create	Create_Item

Standard Parameters

API standard parameters are a collection of parameters that are common to most APIs. The following paragraphs explain the standard parameters that are used in APIs and their interpretation.

Some of the standard parameters apply to all APIs regardless of the nature of the business function they perform. For example, `p_api_version` and `x_return_status` are included in all APIs.

On the other hand, some parameters are applicable for certain types of APIs and not applicable for other types. For example, `p_commit` is applicable for APIs that change the database state, and not applicable for read APIs.

Standard parameters are included in all APIs whenever applicable.

Standard IN parameters:

- `p_api_version`
- `p_init_msg_list`
- `p_commit`
- `p_validation_level`

Standard OUT parameters:

- `x_return_status`
- `x_msg_count`
- `x_msg_data`

Parameter	Type	IN/OUT	Required	Validation
<code>p_api_version</code>	<code>varchar2</code>	IN	Y	Validates version compatibility. The version sent by the calling function is compared to the internal version of the API and an unexpected error (U) is generated if these do not match.
<code>p_init_msg_list</code>	<code>varchar2</code>	IN	N	Used to specify whether the message list can be initialized on entry to the API. It is an optional parameter, and if not supplied, then it defaults to <code>FND_API.G_FALSE</code> , and the API does not initialize the message list.

Parameter	Type	IN/OUT	Required	Validation
p_commit	varchar2	IN	N	Used to specify whether the API can commit its work before returning to the calling function. If not supplied, then it defaults to FND_API.G_FALSE.
x_return_status	varchar2	OUT	-	Specifies whether the API was successful or failed. Valid values are S=successful, E=failed due to expected error, U=failed due to unexpected error.
x_msg_count	number	OUT	-	Specifies number of messages added to message list.
x_msg_data	varchar2	OUT	-	Returns the messages in an encoded format. These messages can then be processed by the standard message functions as defined in Business Object API Coding Standards document.

Value-ID Conversion

IDs are usually used to represent primary and foreign entity keys, and for internal processing of attributes. They are not meaningful and are hidden. Besides IDs, attributes have values that represent them. Those values are meaningful and are used for display purposes. In general, APIs operate only on IDs.

For example, an item is represented by an ID, the number column `item_id`. This ID is its primary key and is used for all internal processing of the item. Besides this ID, an item is represented by a value, the `varchar2` column `item_no`. This value is displayed when you choose an item. Therefore, an item can be identified by either its ID or value, in this case `item_no`.

The following set of rules are for the conversion process:

- Either ID or value, or both can be passed to an API. But, when both values are passed, ID based parameters take precedence over value based parameters. For example, if both parameters are passed, the value based parameter is ignored and the ID based parameter is used.
- When both the value and ID of an attribute are passed to an API, a message informs the API caller that some of the input has been ignored.

- This message is not an error message. The API continues with its regular processing.
- Each value has to resolve into one ID. Failure to resolve a value into an ID is an error and is associated with an error message. The API aborts processing and returns with a return status of error.

Business Objects

This topic describes the parameters needed for each API, as well as the table structure and the associated details.

The topics discussed are:

- Item Create
- Item Lot/Sublot Conversion
- Inventory Quantities
- Lot Create

Item Create

This API is intended as a create function only, used primarily to load item data from legacy systems on implementation. The API does not let item update or unit of measure conversion maintenance. These are handled through additional stored procedures, part of the Item Lot/Sublot Conversion API.

Parameters and Interface

The public Item Create API has the following call interface:

```
GMIPAPI.Create_Item
( p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER   := FND_API.G_VALID_LEVEL_FULL
, p_item_rec         IN  GMIGAPI.item_rec_typ
, x_ic_item_mst_row  OUT ic_item_mst%ROWTYPE
, x_ic_item_cpg_row  OUT ic_item_cpg%ROWTYPE
, x_return_status    OUT  VARCHAR2
, x_msg_count        OUT  NUMBER
, x_msg_data         OUT  VARCHAR2
);
```

If the creation was successful, then the `x_ic_item_mst_row` and `x_ic_item_cpg_row` parameters are returned with the data set up in two tables, regardless of whether it was committed by the procedure.

The contents of the `x_ic_item_cpg` row are undefined if the system constant `SY$CPG_INSTALL` is set to zero, and nothing is written to the `ic_item_cpg` table.

`x_return_status` is returned as `FND_API.G_RET_STS_SUCCESS`, `FND_API.G_RET_STS_UNEXP_ERR`, or `FND_API.G_RET_STS_EXP_ERR`.

The final two parameters returned contain the message count and message stack.

The `p_item_rec` parameter is used to pass the item-specific data required to create an inventory item as described in the following. Refer to the "Item Create API Wrapper" topic for an example of how to populate this parameter and call the stored procedure.

Column	Column Long Name	Len	Default	Req'd	Validation
item_no	item number	32	NULL	Y	Duplicates not allowed on ic_item_mst

Column	Column Long Name	Len	Default	Req'd	Validation
item_desc1	item description 1	70	NULL	Y	Non-blank
item_desc2	item description 2	70	NULL	N	blank
alt_itema	alternative name for item	32	NULL	N	blank
alt_itemb	second alternative name for item	32	NULL	N	blank
item_um	unit of measure	4	NULL	Y	Must exist on sy_uoms_mst
dualum_ind	dual unit of measure indicator	5	NULL	Y	0 = Single UOM control, 1 = Fixed relationship dual UOM control, 2 = Variable relationship dual UOM with default conversion, 3 = Variable relationship dual UOM control
item_um2	secondary unit of measure	4	NULL	N	Mandatory if dual UOM > 0. If non-blank, then must exist on sy_uoms_mst
deviation_lo	factor defining the allowable deviation below the standard conversion for type 2/3	variable	0	N	Must not be negative value. Must be 0 if dualum_ind = 0 or 1
deviation_hi	factor defining the allowable deviation above the standard conversion for type 2/3	variable	0	N	Must not be negative value. Must be 0 if dualum_ind = 0 or 1
level_code	level code	5	0	N	Not currently used
lot_ctl	lot controlled item indicator	5	NULL	Y	0 = Not lot controlled, 1 = Lot controlled
lot_indivisible	item is lot indivisible indicator	5	0	N	0 = Lots are not indivisible, can be split, 1 = Lots are indivisible, cannot be split. Must be 0 if lot_ctl = 0
sublot_ctl	sublot controlled item indicator	5	0	N	0 = Not sub lot controlled, 1 = Sub lot controlled. Must be 0 if lot_ctl = 0

Column	Column Long Name	Len	Default	Req'd	Validation
loct_ctl	location controlled item indicator	5	0	N	0 = Not location controlled, 1 = Location controlled with validation of location, 2 = Location controlled with no validation of location
noninv_ind	non-inventory item indicator	5	0	N	0 = Not a non inventory item - inventory balances maintained, 1 = Non inventory item, inventory balances not maintained. Must be 0 if lot_ctl = 0
match_type	match type	5	3	N	Type of invoice matching done. Blank is no matching, 1 = Invoice only, 2 = Two way matching, 3 = Three way matching
inactive_ind	inactive item indicator	5	0	N	0 = Active, 1 = Inactive
inv_type	inventory type	4	NULL	N	Must exist on ic_invn_typ if non-blank
shelf_life	shelf life	variable	0	N	Must not be negative. Must be 0 if grade_ctl = 0
retest_interval	retest interval	variable	0	N	Must not be negative. Must be 0 if grade_ctl = 0
item_abccode	item ABC code	4	NULL	N	Item ABC code.
gl_class	gl class	8	NULL	N	Must exist on ic_gled_cls if supplied
inv_class	inventory class	8	NULL	N	Must exist on ic_invn_cls if supplied
sales_class	sales class	8	NULL	N	Must exist on ic_sale_cls if supplied
ship_class	ship class	8	NULL	N	Must exist on ic_ship_cls if supplied
frt_class	freight class	8	NULL	N	Must exist on ic_frgt_cls if supplied
price_class	price class	8	NULL	N	Must exist on ic_prce_cls if supplied

Column	Column Long Name	Len	Default	Req'd	Validation
storage_class	storage class	8	NULL	N	Must exist on ic_stor_cls if supplied
purch_class	purchase class	8	NULL	N	Must exist on ic_prch_cls if supplied
tax_class	tax class	8	NULL	N	Must exist on ic_taxn_cls if supplied
customs_class	customs class	8	NULL	N	Must exist on ic_ctms_cls if supplied
alloc_class	allocation class	8	NULL	N	Must exist on ic_allc_cls if supplied
planning_class	planning class	8	NULL	N	Must exist on ps_plng_cls if supplied
itemcost_class	cost class	8	NULL	N	Must exist on ic_cost_cls if supplied
cost_mthd_code	cost method code (not currently used)	4	NULL	N	Must exist on cm_mthd_mst if supplied
upc_code	UPC code (The field is active but not required. It is not validated.)	16	NULL	N	The field is active but not required. It is not validated.
grade_ctl	QC grade controlled item indicator	5	0	N	0 = Not grade controlled, 1 = Grade controlled, Must be 0 if lot_ctl = 0
status_ctl	lot status controlled item indicator	5	0	N	0 = Not status controlled, 1 = Status controlled. Must be 0 if lot_ctl = 0
qc_grade	default QC grade	4	NULL	N	Must exist on qc_grad_mst if non-blank. Must be non-blank if grade_ctl = 1
lot_status	default lot status	4	NULL	N	Must exist on ic_lots_sts if non-blank. Must be non-blank if status_ctl = 1
bulk_id	bulk id (not currently used)	10	0	N	Not currently used
pkg_id	pkg id (not currently used)	10	NULL	N	Not currently used

Column	Column Long Name	Len	Default	Req'd	Validation
qccitem_no	QC reference item	32	NULL	N	Must exist on ic_item_mst if non-blank
qchold_res_code	QC hold reason code	4	NULL	N	Must exist on qc_hrec_mst if non-blank
expaction_code	action code when a lot expires	4	NULL	N	Must exist on qc_actn_mst if non-blank
fill_qty	fill qty (not currently used)	variable	0	N	Not currently used
fill_um	fill um (not currently used)	4	NULL	N	Not currently used
expaction_interval	interval in days between when a lot expires and when the expiry action can be taken	variable	0	N	Must not be negative value
phantom_type	phantom type (not currently used)	5	0	N	Phantom type
whse_item_no	warehouse item number	32	NULL	N	Must exist on ic_item_mst if non-blank
experimental_ind	experimental item indicator	5	NULL	N	0 = Non experimental item, 1 = Experimental item
exported_date	date item was exported to Oracle Financials	variable	01-Jan-1970 + 1 day	N	Exported date
seq_dpnd_class	sequence dependent class	8	NULL	N	Must exist on cr_sqdt_cls if non-blank
commodity_code	commodity code	9	NULL	N	Must exist on ic_comd_cds if non-blank. Must be non-blank if GMI:Intrastat= "1"
ic_matr_days	lot maturity days	variable	0	N	Must not be negative value
ic_hold_days	lot hold days	variable	0	N	Must not be negative value
user_name	user name	100	OPM	N	Ignored, but retained for backward compatibility.
attribute1	attribute1	240	NULL	N	Descriptive flexfield segment
attribute2	attribute2	240	NULL	N	Descriptive flexfield segment
attribute3	attribute3	240	NULL	N	Descriptive flexfield segment

Column	Column Long Name	Len	Default	Req'd	Validation
attribute4	attribute4	240	NULL	N	Descriptive flexfield segment
attribute5	attribute5	240	NULL	N	Descriptive flexfield segment
attribute6	attribute6	240	NULL	N	Descriptive flexfield segment
attribute7	attribute7	240	NULL	N	Descriptive flexfield segment
attribute8	attribute8	240	NULL	N	Descriptive flexfield segment
attribute9	attribute9	240	NULL	N	Descriptive flexfield segment
attribute10	attribute10	240	NULL	N	Descriptive flexfield segment
attribute11	attribute11	240	NULL	N	Descriptive flexfield segment
attribute12	attribute12	240	NULL	N	Descriptive flexfield segment
attribute13	attribute13	240	NULL	N	Descriptive flexfield segment
attribute14	attribute14	240	NULL	N	Descriptive flexfield segment
attribute15	attribute15	240	NULL	N	Descriptive flexfield segment
attribute16	attribute16	240	NULL	N	Descriptive flexfield segment
attribute17	attribute17	240	NULL	N	Descriptive flexfield segment
attribute18	attribute18	240	NULL	N	Descriptive flexfield segment
attribute19	attribute19	240	NULL	N	Descriptive flexfield segment
attribute20	attribute20	240	NULL	N	Descriptive flexfield segment
attribute21	attribute21	240	NULL	N	Descriptive flexfield segment
attribute22	attribute22	240	NULL	N	Descriptive flexfield segment
attribute23	attribute23	240	NULL	N	Descriptive flexfield segment
attribute24	attribute24	240	NULL	N	Descriptive flexfield segment
attribute25	attribute25	240	NULL	N	Descriptive flexfield segment
attribute26	attribute26	240	NULL	N	Descriptive flexfield segment
attribute27	attribute27	240	NULL	N	Descriptive flexfield segment
attribute28	attribute28	240	NULL	N	Descriptive flexfield segment
attribute29	attribute29	240	NULL	N	Descriptive flexfield segment
attribute30	attribute30	240	NULL	N	Descriptive flexfield segment

Column	Column Long Name	Len	Default	Req'd	Validation
attribute_ category	attribute category	30	NULL	N	Descriptive flexfield structure defining column
ont_pricing_qty_ source	number	5	NULL	Y	OPM/OM integration pricing by quantity 2 indicator. Populated with 0 (zero) to indicate pricing by order quantity or 1 to indicate pricing by secondary quantity.

Item Lot/Sublot Conversion

This API is intended as a create function only. The create function is used primarily to load item data from legacy systems on implementation.

Item Lot/Sublot Conversion API - Parameters

There are two variants of this procedure that have the same name, but reside in different packages and have different signatures.

The public procedure has the following call interface:

GMIPAPI.Create_Item_Lot_Conv

```
(
  p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER    := FND_API.G_VALID_LEVEL_FULL
, p_conv_rec         IN  GMIPAPI.conv_rec_typ
, x_ic_item_cnv_row  OUT  ic_lots_mst%ROWTYPE
, x_return_status    OUT  VARCHAR2
, x_msg_count        OUT  NUMBER
, x_msg_data         OUT  VARCHAR2
);
```

The first four, and last three parameters are standard across all of the API calls and are identical to the Create Item API. If the creation is successful, then the `x_ic_item_cnv_row` parameter is returned with the data set up in the table, regardless of whether it was committed by the procedure.

The group procedure has the following call interface:

GMIGAPI.Create_Item_Lot_Conv

```
(
  p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER    := FND_API.G_VALID_LEVEL_FULL
, p_conv_rec         IN  GMIGAPI.conv_rec_typ
, p_ic_item_mst_row  IN  ic_item_mst%ROWTYPE
, p_ic_lots_mst_row  IN  ic_lots_mst%ROWTYPE
, x_ic_item_cnv_row  OUT  ic_lots_mst%ROWTYPE
, x_return_status    OUT  VARCHAR2
```

```

, x_msg_count      OUT NUMBER
, x_msg_data       OUT VARCHAR2
);

```

This procedure takes two additional parameters compared to the P variant for use when item and lot data are already known. Then p_ic_item_mst_row and p_ic_lots_mst_row are passed with the appropriate data. This can be found by calling the GMIGUTL.Get_Item and GMIGUTL.Get_Lot procedures. All other IN and OUT parameters are identical to the public API.

Column	Column Long Name	Type	Len	Default	Req'd	Validation
item_no	item number	varchar2	32	NULL	Y	Must exist on ic_item_mst, must not be deleted, and must be active
lot_no	lot number	varchar2	32	NULL	N	Must be blank if ic_item_mst.lot_ctl = 0. If ic_item_mst.sublot_ctl = 0, then item_no+lot_no must exist in ic_lots_mst
sublot_no	sublot number	varchar2	32	NULL	N	Must be blank if ic_item_mst.sublot_ctl = 0. Must be blank if lot_no is blank. If supplied, then item_no+lot_no+sublot_no must exist on ic_lots_mst
from_uom	from unit of measure	varchar2	4	NULL	Y	Must exist on sy_uoms_mst. Must be of same UOM type as ic_item_mst.item_um
to_uom	to unit of measure	varchar2	4	NULL	Y	Must exist on sy_uoms_mst. Must be of different UOM type to ic_item_mst.item_um
type_factor	conversion factor	number		NULL	Y	Conversion factor. 1 from-uom equals this number of to_uom
user_name	user name	varchar2	100	OPM	N	Ignored but retained for backward compatibility

Inventory Quantities

This API deals with the following functions within the OPM Inventory application:

- Create inventory
- Adjust inventory
- Move inventory
- Change lot status
- Change QC grade

Inventory Quantities - Parameters

There are two variants of this procedure-public and group. They have the same name, but are distinguished by residing in separate packages, and have different signatures. The public call interface is:

GMIPAPI.Inventory_Posting

```
(
  p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER    := FND_API.G_VALID_LEVEL_FULL
, p_qty_rec          IN  GMIPAPI.qty_rec_tpy
, x_ic_jrnl_mst_row  OUT  ic_jrnl_mst%ROWTYPE
, x_ic_adjs_jnl_row1 OUT  ic_adjs_jnl%ROWTYPE
, x_ic_adjs_jnl_row2 OUT  ic_adjs_jnl%ROWTYPE
, x_return_status    OUT  VARCHAR2
, x_msg_count        OUT  NUMBER
, x_msg_data         OUT  VARCHAR2
);
```

The first four, and last three parameters are standard across all of the API calls and are identical to the Create Item API. If the posting is successful, then the `x_ic_jrnl_mst_row` and `x_ic_adjs_jnl_row` parameters are returned with the data set up in the tables, regardless of whether it was committed by the procedure. For inventory movements, grade, and status changes `x_ic_adjs_jnl_row1` contains the from and `x_ic_adjs_jnl_row2` contains the to. For all other calls, the `x_ic_adjs_jnl_row2` returns undefined.

The group level call interface is:

GMIGAPI.Inventory_Posting

```
(
  p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER    := FND_API.G_VALID_LEVEL_FULL
, p_qty_rec          IN  GMIGAPI.qty_rec_typ
, p_ic_item_mst_row  IN  ic_item_mst%ROWTYPE
, p_ic_item_cpg_row  IN  ic_item_cpg%ROWTYPE
, p_ic_lots_mst_row  IN  ic_lots_mst%ROWTYPE
, p_oc_lots_cpg_row  IN  ic_lots_cpg%ROWTYPE
, x_ic_jrnl_mst_row  OUT  ic_jrnl_mst%ROWTYPE
, x_ic_adjs_jnl_row1 OUT  ic_adjs_jnl%ROWTYPE
, x_ic_adjs_jnl_row2 OUT  ic_adjs_jnl%ROWTYPE
, x_return_status    OUT  VARCHAR2
, x_msg_count        OUT  NUMBER
, x_msg_data         OUT  VARCHAR2
);
```

This version takes in extra parameters for use when the appropriate data is known in the calling program in the same way as above.

Column	Column Long Name	Type	Len	Default	Req'd	Validation
trans_type	transaction type	number	2	NULL	Y	Valid values are: 1 - create inventory, 2 - adjust inventory, 3 - move inventory, 4 - change lot status, 5 - change QC grade. No other values allowed
item_no	item number	varchar2	32	NULL	Y	Must exist on ic_item_mst. Must not be deleted. Can be inactive if the IC\$ALLOW_INACTIVE flag is set to 1, must not be non inventory
journal_no	journal number	varchar2	32	NULL	N	Must be blank if automatic document sequencing. Must not exist in ic_jrnl_mst if manual document sequencing
from_whse_code	from warehouse code	varchar2	4	NULL	Y	May be blank if trans_type = 5. Must exist on ic_whse_mst for all other transaction types

Column	Column Long Name	Type	Len	Default	Req'd	Validation
to_whse_code	to warehouse code	varchar2	4	NULL	N	Must exist on ic_whse_mst if trans_type = 3
item_um	primary unit of measure	varchar2	4	NULL	Y	Must exist on sy_uoms_mst. Must be of same UOM type as primary UOM of item
item_um2	secondary unit of measure	varchar2	4	NULL	Y	Must exist on sy_uoms_mst. Must be of same UOM type as secondary UOM of item
lot_no	lot number	varchar2	32	blank	N	Must be blank if ic_item_mst.lot_ctl = 0. Must not equal GMI:Default Lot
sublot_no	sublot number	varchar2	32	blank	N	Must be blank if ic_item_mst.sublot_ctl = 0
from_location	from location	varchar2	16	NULL	N	From location
to_location	to location	varchar2	16	NULL	N	To location
trans_qty	primary transaction quantity	number	variable	NULL	N	Transaction quantity
trans_qty2	secondary transaction quantity	number		NULL	N	Transaction quantity 2
qc_grade	QC grade	varchar2	4	NULL	N	Must exist on qc_grad_mst if supplied
lot_status	lot status	varchar2	4	NULL	N	Must exist on ic_lots_mst if supplied
co_code	company code	varchar2	4	NULL	Y	Must exist in sy_orgn_mst
orgn_code	organization code	varchar2	4	NULL	N	Must exist in sy_orgn_mst and sy_orgn_mst.co_code must be co_code above
trans_date	transaction date	date		NULL	Y	Must be in an open inventory calendar
reason_code	reason code	varchar2	4	NULL	Y	Must exist in sy_reas_cds
user_name	user name	varchar2	100	'OPM'	N	Ignored but retained for backward compatibility

Lot Create

This API is intended as a create function only. The create function is used primarily to load item data from legacy systems on implementation. The Lot Create API does not let lot update, or delete.

Lot Create API - Parameters

There are two variants of this procedure-public and group. They have the same name but reside in separate packages and have different signatures:

The public call interface is:

GMIPAPI.Create_Lot

```
( p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER    := FND_API.G_VALID_LEVEL_FULL
, p_lot_rec          IN  GMIPAPI.lot_rec_typ
, x_ic_lots_mst_rec  OUT  ic_lots_mst%ROWTYPE
, x_ic_lots_cpg_rec  OUT  ic_lots_cpg%ROWTYPE
, x_return_status    OUT  VARCHAR2
, x_msg_count        OUT  NUMBER
, x_msg_data         OUT  VARCHAR2
);
```

The first four, and last three parameters are standard across all of the API calls and are identical to the Create Item API. If the lot creation is successful, then `x_ic_lots_mst_row` and `x_ic_lots_cpg_row` parameters are returned with the data set up in the two tables, regardless of whether it was committed by the procedure.

If the `SY$CPG_INSTALL` flag is set to zero, then the contents of the row returned in `x_ic_lots_cpg` are undefined and nothing is written to the `ic_lots_cpg` table.

The group version is as follows:

GMIGAPI.Create_Lot

```
( p_api_version      IN  NUMBER
, p_init_msg_list    IN  VARCHAR2 := FND_API.G_FALSE
, p_commit           IN  VARCHAR2 := FND_API.G_FALSE
, p_validation_level IN  NUMBER    := FND_API.G_VALID_LEVEL_FULL
, p_lot_rec          IN  GMIGAPI.lot_rec_typ
, p_ic_item_mst_row  IN  ic_item_mst%ROWTYPE
, p_ic_item_cpg_row  IN  ic_item_cpg%ROWTYPE
, x_ic_lots_mst_rec  OUT  ic_lots_mst%ROWTYPE
```

```

, x_ic_lots_cpg_rec OUT ic_lots_cpg%ROWTYPE
, x_return_status OUT VARCHAR2
, x_msg_count OUT NUMBER
, x_msg_data OUT VARCHAR2
);

```

This procedure takes two additional parameters compared to the P variant for use when item data is already known. If this is the case, then p_ic_item_mst_row and p_ic_item_mst_cpg are passed with the appropriate data. This can be found by calling the GMIGUTL.Get_Item procedure. All other IN and OUT parameters are identical to the public API, and the GMA:CPG INSTALL flag are treated in the same way.

Column	Column Long Name	Type	Len	Default	Req'd	Validation
item_no	item number	varchar1	32	NULL	Y	Must exist ic_item_mst. Must not be deleted. Must be active. Must be lot-controlled
lot_no	lot number	varchar2	32	NULL	Y	Non-blank
sublot_no	sublot number	varchar2	32	NULL	Y	Must be blank if ic_item_mst.sublot_ctl = 0. The item_no/lot_no/sublot_no must not already exist on ic_lots_mst
lot_desc	lot description	varchar2	32	NULL	N	Lot description
qc_grade	p_init_msg_list	varchar2	40	NULL	N	Must be blank if ic_item_mst.grade_ctl = 0. Must not be blank if ic_item_mst.grade_ctl = 1. Must exist on qc_grad_mst if non-blank
expaction_code	expiry action code	varchar2	4	ic_item_mst.qc_grade	N	Must exist on qc_actn_mst if non-blank
expaction_date	expiry action date	varchar2	4	ic_item_mst.expaction_code	N	Must not be less than lot_created date. Must not be less than expiry_date

Column	Column Long Name	Type	Len	Default	Req'd	Validation
lot_created	lot created date	date	variable	IF ic_item_ mst.grade_ctl = 1AND ic_ item_ mst.expection _interval > 0 THEN expire_ date + ic_ item_ mst.expection _interval ELSE 31-Dec-2010 00:00:00 (SY\$MAX_ DATE)	N	Lot created
expire_date	lot expire date	date	variable	System Date	N	Must not be less than lot_created date
retest_date	retest date	date	variable	IF ic_item_ mst.grade_ctl = 1AND ic_ item_ mst.shelf_life > 0 THEN lot_ created + ic_ item_ mst.shelf_life ELSE 31-Dec-2010 00:00:00 (SY\$MAX_ DATE)	N	Must not be less than lot_created date

Column	Column Long Name	Type	Len	Default	Req'd	Validation
strength	strength	date	variable	IF ic_item_mst.grade_ctl = 1 AND ic_item_mst.retest_interval > 0 THEN lot_created + ic_item_mst.retest_interval ELSE 31-Dec-2010 00:00:00 (SY\$MAX_DATE)	N	Must be zero or positive value
inactive_ind	inactive indicator	number	variable	100	N	Must be 0 or 1
origination_type	origination type	number	5	0	N	Must exist on sy_type_mst where table_name = ic_lots_mst and field_name = 'origination_type'
shipvendor_no	shipping vendor number	number	5	0	N	Must exist on po_vend_mst if non-blank
vendor_lot_no	vendor lot number	varchar2	32	NULL	N	Vendor lot number.
ic_matr_date	item maturity date	varchar2	32	NULL	N	Item maturity date.
ic_hold_date	item hold date	date	variable	lot_created + ic_item_mst.ic_matr_days	N	Item hold date.
user_name	user name	varchar2	100	OPM	Y	Any valid user name.
attribute1	attribute1	varchar2	240	NULL	N	Descriptive flexfield segment
attribute2	attribute2	varchar2	240	NULL	N	Descriptive flexfield segment
attribute3	attribute3	varchar2	240	NULL	N	Descriptive flexfield segment

Column	Column Long Name	Type	Len	Default	Req'd	Validation
attribute4	attribute4	varchar2	240	NULL	N	Descriptive flexfield segment
attribute5	attribute5	varchar2	240	NULL	N	Descriptive flexfield segment
attribute6	attribute6	varchar2	240	NULL	N	Descriptive flexfield segment
attribute7	attribute7	varchar2	240	NULL	N	Descriptive flexfield segment
attribute8	attribute8	varchar2	240	NULL	N	Descriptive flexfield segment
attribute9	attribute9	varchar2	240	NULL	N	Descriptive flexfield segment
attribute10	attribute10	varchar2	240	NULL	N	Descriptive flexfield segment
attribute11	attribute11	varchar2	240	NULL	N	Descriptive flexfield segment
attribute12	attribute12	varchar2	240	NULL	N	Descriptive flexfield segment
attribute13	attribute13	varchar2	240	NULL	N	Descriptive flexfield segment
attribute14	attribute14	varchar2	240	NULL	N	Descriptive flexfield segment
attribute15	attribute15	varchar2	240	NULL	N	Descriptive flexfield segment
attribute16	attribute16	varchar2	240	NULL	N	Descriptive flexfield segment
attribute17	attribute17	varchar2	240	NULL	N	Descriptive flexfield segment
attribute18	attribute18	varchar2	240	NULL	N	Descriptive flexfield segment
attribute19	attribute19	varchar2	240	NULL	N	Descriptive flexfield segment
attribute20	attribute20	varchar2	240	NULL	N	Descriptive flexfield segment

Column	Column Long Name	Type	Len	Default	Req'd	Validation
attribute21	attribute21	varchar2	240	NULL	N	Descriptive flexfield segment
attribute22	attribute22	varchar2	240	NULL	N	Descriptive flexfield segment
attribute23	attribute23	varchar2	240	NULL	N	Descriptive flexfield segment
attribute24	attribute24	varchar2	240	NULL	N	Descriptive flexfield segment
attribute25	attribute25	varchar2	240	NULL	N	Descriptive flexfield segment
attribute26	attribute26	varchar2	240	NULL	N	Descriptive flexfield segment
attribute27	attribute27	varchar2	240	NULL	N	Descriptive flexfield segment
attribute28	attribute28	varchar2	240	NULL	N	Descriptive flexfield segment
attribute29	attribute29	varchar2	240	NULL	N	Descriptive flexfield segment
attribute30	attribute30	varchar2	240	NULL	N	Descriptive flexfield segment
attribute_category	attribute_category	varchar2	240	NULL	N	Descriptive flexfield structure definition

Messages and Errors

This appendix covers:

- Handling Messages
- Interpreting Error Conditions
- Understanding Error Messages

Handling Messages

APIs put result messages into a message list. Programs calling APIs can then get the messages from the list and process them by issuing them, loading them in a database table, or writing them to a log file.

Messages are stored in an encoded format to enable API callers to find out message names by using the standard functions provided by the message dictionary. It also allows storing these messages in database tables and reporting off these tables in different languages.

The structure of the message list is not public. Neither API developers nor API callers can access this list except through calling the API message utility routines mentioned below.

The following utility functions are defined in the FND_MSG_PUB package, in the file AFASMSG.S.pls:

Initialize Initializes the API message list.

Add Adds a message to the API message list.

Get Gets a message from the API message list.

Count_Msg Returns the number of messages in the API message list.

Delete Deletes one or more messages from the API message list.

Reset Resets the index used in getting messages.

Count_And_Get Returns the number of messages in the API message list. If this number is one, then it also returns the message data.

Refer to the documentation of these functions and procedures for usage information.

To add a message to the API message list, developers should use the regular message dictionary procedures `FND_MESSAGE.SET_NAME` and `FND_MESSAGE.SET_TOKEN` to set the message name and tokens on the message dictionary stack. They should then call `FND_MSG_PUB.Add` to fetch the messages off the message dictionary stack and add it to the API message list.

To get a message from the API message list, API callers should use the procedure `FND_MSG_PUB.Get`. This procedure operates in 5 different modes:

First Gets the first message in the API message list.

Next Gets the next message in the API message list.

Last Gets the last message in the API message list.

Previous Gets the previous message in the API message list.

Specific Gets a specific message from the API message list.

For better performance and reduction in the overall number of calls a program needs to make in order to execute an API, it is recommended that APIs provide their callers with the following information:

- message count
- message data

The message count holds the number of messages in the API message list. If this number is one, then message data holds the message in an encoded format.

Interpreting Error Conditions

The parameter `x_return_status` indicates whether the API was successful or failed. The values are as follows:

- S for success
- E for error
- U unknown or unexpected status

Understanding Error Messages

These error messages are output to the stored procedure message file, and can be monitored through the return `x_msg_count`. In conjunction with the `x_return_status`, this can be used to monitor the success or failure of the procedure call.

Displaying Errors in Languages Other than English

Language translation of error messages is determined by the environment variable `NLS_LANGUAGE`. If the message is not found in the required language, then the message is retrieved in US English.

The following is a complete listing of the Inventory API Error Messages. Note that a message that is preceded with Warning is not a fatal API error, just a warning, and a message preceded with Error is a fatal API error.

Any uppercase word preceded by an ampersand (&) is a token, or placeholder, for an actual value that will be populated at runtime.

Message Code	Message Name
IC_API_ITEM_ALREADY_EXISTS	Item number &ITEM already exists
IC_API_INVALID_UOM	Invalid unit of measure &UOM for item number &ITEM
IC_API_INVALID_DUALUM_IND	Dual unit of measure indicator not in range 0 - 3 for item number &ITEM
IC_API_INVALID_DEVIATION	Invalid deviation factor for item number &ITEM
IC_API_INVALID_LOT_CTL	Invalid lot control flag for item number &ITEM
IC_API_INVALID_LOT_INDIVISIBLE	Invalid lot indivisible flag for item number &ITEM

Message Code	Message Name
IC_API_INVALID_SUBLLOT_CTL	Invalid subplot control flag for item number &ITEM
IC_API_INVALID_LOCT_CTL	Invalid location control flag for item number &ITEM
IC_API_INVALID_NONINV_IND	Invalid non-inventory flag for item number &ITEM
IC_API_INVALID_MATCH_TYPE	Invalid match type for item number &ITEM
IC_API_INVALID_INV_TYPE	Invalid inventory type for item number &ITEM
IC_API_INVALID_INACTIVE_IND	Invalid inactive flag for item number &ITEM
IC_API_INVALID_SHELF_LIFE	Invalid shelf life for item number &ITEM
IC_API_INVALID_RETEST_INTERVAL	Invalid retest interval for item number &ITEM
IC_API_INVALID_ABCCODE	Invalid ABC code for item number &ITEM
IC_API_INVALID_GL_CLASS	Invalid GL class for item number &ITEM
IC_API_INVALID_INV_CLASS	Invalid inventory class for item number &ITEM
IC_API_INVALID_SALES_CLASS	Invalid sales class for item number &ITEM
IC_API_INVALID_SHIP_CLASS	Invalid ship class for item number &ITEM
IC_API_INVALID_FRT_CLASS	Invalid freight class for item number &ITEM
IC_API_INVALID_PRICE_CLASS	Invalid price class for item number &ITEM
IC_API_INVALID_STORAGE_CLASS	Invalid storage class for item number &ITEM
IC_API_INVALID_PURCH_CLASS	Invalid purchase class for item number &ITEM
IC_API_INVALID_TAX_CLASS	Invalid tax class for item number &ITEM
IC_API_INVALID_CUSTOMS_CLASS	Invalid customs class for item number &ITEM
IC_API_INVALID_ALLOC_CLASS	Invalid allocation class for item number &ITEM
IC_API_INVALID_PLANNING_CLASS	Invalid planning class for item number &ITEM
IC_API_INVALID_ITEMCOST_CLASS	Invalid item cost class for item number &ITEM
IC_API_INVALID_COST_MTHD_CODE	Invalid cost method for item number &ITEM
IC_API_INVALID_GRADE_CTL	Invalid grade control flag for item number &ITEM
IC_API_INVALID_STATUS_CTL	Invalid status control flag for item number &ITEM

Message Code	Message Name
IC_API_INVALID_QC_GRADE	Invalid QC grade for item number &ITEM
IC_API_INVALID_LOT_STATUS_API	Invalid lot status for item number &ITEM
IC_API_INVALID_QCITEM_NO	Invalid QC reference item number for item number &ITEM
IC_API_INVALID_QCHOLD_RES_CODE	Invalid QC hold reason code for item number &ITEM
IC_API_INVALID_EXPACTION_CODE	Invalid expiry action code for item number &ITEM
IC_API_INVALID_WHSE_ITEM_NO	Invalid warehouse item number for item number &ITEM
IC_API_INVALID_EXPERIMENTAL_IND	Invalid experimental indicator for item number &ITEM
IC_API_INVALID_SEQ_DPND_CLASS	Invalid sequence dependent class for item number &ITEM
IC_API_INVALID_COMMODITY_CODE	Invalid commodity code for item number &ITEM
IC_API_INVALID_MATR_DAYS	Invalid maturity days for item number &ITEM
IC_API_INVALID_HOLD_DAYS	Invalid hold release days for item number &ITEM
SY_API_UNABLE_TO_GET_SURROGATE	Failed to get &SKEY surrogate key
IC_API_INVALID_ITEM_NO	Invalid item &ITEM_NO
IC_API_INVALID_LOT_NO	Invalid lot &LOT_NO - &SUBLOT_NO for item &ITEM_NO
IC_API_INVALID_LOT_UOM_TYPE	Invalid UOM type for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO UOM &UOM
IC_API_INVALID_LOT_UOM	Invalid UOM &UOM for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO
IC_API_ITEM_CNV_ALREADY_EXISTS	Conversion already exists for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO UOM type &UM_TYPE
IC_API_INVALID_TYPE_FACTOR	Invalid conversion factor for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO

Message Code	Message Name
IC_API_LOT_ITEM_UOM_MISMATCH	From UOM is of wrong type for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO
IC_API_ITEM_LOT_UOM_FAILED	Cannot convert from &UM1 to &UM2 for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO
IC_API_INVALID_TRANS_TYPE	Invalid transaction type &TRANS_TYPE
IC_API_INVALID_JOURNAL_NO	Invalid journal number &JOURNAL_NO
SY_API_UNABLE_TO_GET_DOC_NO	Failed to get doc number for type &DOC_TYPE organization &ORGN_CODE
IC_API_INVALID_ITEM_NO	Invalid item &ITEM_NO
IC_API_TRANS_TYPE_FOR_ITEM	Invalid transaction type &TRANS_TYPE for item &ITEM_NO
IC_API_INVALID_LOT_NO	Invalid lot &LOT_NO - &SUBLOT_NO for item &ITEM_NO
IC_API_INVALID_UOM	Invalid unit of measure &UOM for item number &ITEM_NO
IC_API_SUBLOT_NOT_REQD	Sub-lot is not required for item &ITEM_NO as it is not sub-lot controlled
IC_API_INVALID_WHSE_CODE	Invalid warehouse code &WHSE_CODE
IC_API_INVALID_LOCATION	Invalid location &LOCATION warehouse code &WHSE_CODE
IC_API_INVALID_QTY	Invalid quantities for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO
IC_API_LOCT_ONHAND_EXISTS	Inventory exists item &ITEM_NO lot &LOT_NO - &SUBLOT_NO at &WHSE_CODE-&LOCATION
IC_API_NO_LOCT_ONHAND	No inventory for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO at &WHSE_CODE-&LOCATION
IC_API_NEG_QTY_NOT_ALLOWED	Negative qty - item &ITEM_NO lot &LOT_NO - &SUBLOT_NO at &WHSE_CODE-&LOCATION
IC_API_MOVE_STATUS_ERR	Move status error for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO
IC_API_INVALID_LOT_STATUS	Invalid lot status for item number &ITEM_NO

Message Code	Message Name
IC_API_INVALID_QC_GRADE	Invalid QC grade for item number &ITEM_NO
SY_API_INVALID_REASON_CODE	Invalid reason code &REASON_CODE
SY_API_INVALID_CO_CODE	Invalid company code &CO_CODE
SY_API_INVALID_ORGN_CODE	Invalid organization code &ORGN_CODE
IC_API_INVALID_TRANS_DATE	Invalid transaction date &TRANS_DATE
SY_API_UNABLE_TO_GET_SURROGATE	Failed to get &SKEY surrogate key
SY_API_INVALID_OP_CODE	Invalid operator code &OP_CODE
IC_API_INVALID_ITEM	Invalid item &ITEM
IC_API_ITEM_NOT_LOT_CTL	Item &ITEM is not lot controlled
IC_API_SUBLLOT_NOT_REQD	Sub-lot &SUBLLOT supplied for item &ITEM, lot &LOT which is not subplot controlled
IC_API_LOT_ALREADY_EXISTS	Lot &LOT - &SUBLLOT for item number &ITEM already exists
IC_API_INVALID_LOT_QC_GRADE	Invalid QC grade for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_LOT_EXPIACTION_CODE	Invalid expiry action for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_EXPIACTION_DATE_API	Invalid expiry action date for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_EXPIRE_DATE	Invalid expire date for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_RETEST_DATE	Invalid retest date for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_LOT_STRENGTH	Invalid strength for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_LOT_INACTIVE_IND	Invalid inactive indicator for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_LOT_ORIGINATION_TYPE	Invalid origination type for item &ITEM lot &LOT - &SUBLLOT
IC_API_INVALID_LOT_SHIPVENDOR_NO	Invalid ship vendor for item &ITEM lot &LOT - &SUBLLOT

Message Code	Message Name
IC_API_INVALID_LOT_MATR_DAYS	Invalid maturity days for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_HOLD_DAYS	Invalid hold days for item &ITEM lot &LOT - &SUBLOT
SY_API_UNABLE_TO_GET_SURROGATE	Failed to get &SKEY surrogate key

File Error Messages

These messages are related to the handling of the ASCII flat input and output files. They are sent to the standard output device as it is inappropriate to attempt to send them to the files which themselves can cause the erroneous condition. They are hard-coded in English.

Message Name	Message Code
UTL_FILE.INVALID_OPERATION	Invalid operation for 'FILE'
UTL_FILE.INVALID_PATH	Invalid path for 'FILE'
UTL_FILE.INVALID_MODE	Invalid mode for 'FILE'
UTL_FILE.INVALID_FILEHANDLE	Invalid File handle for 'FILE'
UTL_FILE.WRITE_ERROR	Invalid Write Error for 'FILE'
UTL_FILE.READ_ERROR	Invalid Read Error for 'FILE'
UTL_FILE.INTERNAL_ERROR	Internal Error

How to Get Your OPM Inventory APIs Running

This appendix is used in conjunction with the rest of the guide, but is not intended to replace it. The information below is supplemental material that provides additional information regarding OPM Inventory APIs.

Following are four steps to use the Inventory APIs:

1. Create a wrapper file
2. Create a parameter file
3. Determine where log files writes and open permissions
4. Run the API

Step 1 Creating the Wrapper File

Below are some actual wrapper files that you can copy, use, and modify to get these APIs working.

Valid Username

When running the API there has to be a valid username for logging in to create the item or transaction. This username must be a valid applications username, and must be put in the following line of code in the wrapper file. In this example, PROCESS_OPS is a valid applications user.

```
return_sts :=gmigutl.setup('PROCESS_OPS');
```

Call to gmigutl.setup

It is important that your wrapper make a call to the gmigutl.setup procedure. This is new to 11i, and this call is necessary in order for the API to complete successfully.

Example Create Item Wrapper

Copy wrapper below to file.

```
set serveroutput on size 2000
set timing on
DECLARE
return_sts BOOLEAN;
tester NUMBER;
message VARCHAR2(240);
l_return_code VARCHAR2(5);
BEGIN
return_sts :=gmigutl.setup('PROCESS_OPS');
if return_sts = true then
l_return_code :='TRUE';
dbms_output.put_line('Setup is ' ||l_return_code);
gmi_item_wrp.CREATE_ITEM('/usr/tmp','icitem.txt','icitem.log',',');
dbms_output.put_line('Create itemAPI has completed');
dbms_output.put_line('Please check the log files to make sure no errors
occurred during execution');
else
l_return_code :='FALSE';
dbms_output.put_line('Setup is ' || l_return_code);
fnd_msg_pub.get
(p_msg_index => 1,
p_data => message,
p_encoded => FND_API.G_FALSE,
p_msg_index_out => tester
);
dbms_output.put_line(message);
end if;
END;
/
set serveroutput off
set timing off
```

Note: To use this wrapper, ensure you modify the following lines, making the changes highlighted in the earlier sections.

```
return_sts :=gmigutl.setup('PROCESS_OPS');
gmi_item_wrp.CREATE_ITEM('/usr/tmp','icitem.txt','icitem.log',',');
```

Example Quantities API

Copy wrapper below to file.

```
set serveroutput on size 2000
set timing on
DECLARE
return_sts BOOLEAN;
tester NUMBER;
message VARCHAR2(240);
l_return_code VARCHAR2(5);
BEGIN
return_sts :=gmitutl.setup('PROCESS_OPS');
if return_sts = true then
l_return_code :='TRUE';
dbms_output.put_line('Setup is ' ||l_return_code);
GMI_QUANTITY_WRP.post ('/usr/tmp','icqty.txt','icqty.log','');
dbms_output.put_line('Quantities API has completed');
dbms_output.put_line('Please check the log files to make sure no errors
occurred during execution');
else
l_return_code :='FALSE';
dbms_output.put_line('Setup is ' || l_return_code);
fnd_msg_pub.get
(p_msg_index => 1,
p_data => message,
p_encoded => FND_API.G_FALSE,
p_msg_index_out => tester
);
dbms_output.put_line(message);
end if;
END;
/
set serveroutput off
set timing off
```

Note: To use this wrapper, ensure you modify the following lines, making the changes highlighted in the earlier sections.

```
return_sts :=gmitutl.setup('PROCESS_OPS');
GMI_QUANTITY_WRP.post ('/usr/tmp','icqty.txt','icqty.log','');
```

Lot Create API

Copy wrapper below to file.

```
set serveroutput on size 2000
set timing on
DECLARE
return_sts BOOLEAN;
tester NUMBER;
message VARCHAR2(240);
l_return_code VARCHAR2(5);
BEGIN
return_sts :=gmigutl.setup('PROCESS_OPS');
if return_sts = true then
l_return_code :='TRUE';
dbms_output.put_line('Setup is ' ||l_return_code);
GMI_LOTS_WRP.Create_Lot('/usr/tmp','iclot.txt','iclot.log','');
dbms_output.put_line('Lot Create API has completed');
dbms_output.put_line('Please check the log files to make sure no errors
occurred during execution');
else
l_return_code :='FALSE';
dbms_output.put_line('Setup is ' || l_return_code);
fnd_msg_pub.get
(p_msg_index => 1,
p_data => message,
p_encoded => FND_API.G_FALSE,
p_msg_index_out => tester
);
dbms_output.put_line(message);
end if;
END;
/
set serveroutput off
set timing off
```

Note: To use this wrapper, ensure you modify the following lines, making the changes highlighted in the earlier sections.

```
return_sts :=gmigutl.setup('PROCESS_OPS');
GMI_LOTS_WRP.Create_Lot ('/usr/tmp','iclot.txt','iclot.log','');
```

Example Unit Of Measure API

Copy wrapper below to file.

```
set serveroutput on size 2000
set timing on
DECLARE
return_sts BOOLEAN;
tester NUMBER;
message VARCHAR2(240);
l_return_code VARCHAR2(5);
BEGIN
return_sts :=gmigutl.setup('PROCESS_OPS');
if return_sts = true then
l_return_code :='TRUE';
dbms_output.put_line('Setup is ' ||l_return_code);
GMI_ITEM_LOT_CONV_WRP.create_conv('/usr/tmp','icuum.txt','icuum.log','');
dbms_output.put_line('Create UOM API has completed');
dbms_output.put_line('Please check the log files to make sure no errors
occurred during execution');
else
l_return_code :='FALSE';
dbms_output.put_line('Setup is ' || l_return_code);
fnd_msg_pub.get
(p_msg_index => 1,
p_data => message,
p_encoded => FND_API.G_FALSE,
p_msg_index_out => tester
);
dbms_output.put_line(message);
end if;
END;
/
set serveroutput off
set timing off
```

Note: To use this wrapper, ensure you modify the following lines, making the changes highlighted in the earlier sections.

```
return_sts :=gmigutl.setup('PROCESS_OPS');
GMI_ITEM_LOT_CONV_WRP.create_conv('/usr/tmp','icuum.txt','icuum.log','');
```

Step 2 Parameter File

The parameter file is a text file that lists all parameters depending on which particular API you are running, and what actions you are attempting to do.

This parameter file must be placed in the same directory as the previous log file, or the syntax in the above line of code would have to change to tell the API where to look for it.

Below is a list of the parameters required by each of the APIs. In addition, there is a copy of one parameter file that was used to create an item to see how the file should look. For non-required fields that you want to be blank, use the comma to hold the place. This is an example of what your parameter file should look like for creating an item.

Example

```
API100,newitem,100,,,LB,0,,,,,0,0,0,0,0,,0,,,,,,,,,,,,,,,,,,,,,0,0,,,,,,,,,,,,,API
100,0,,,NONE,,,PROCESS_OPS,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

Item Create API

```
item_no, item_desc1, item_desc2, alt_itema, alt_itemb, item_um, dualum_ind,
item_um2, deviation_lo, deviation_hi, level_code, lot_ctl, lot_indivisible,
sublot_ctl, loct_ctl, noninv_ind, match_type, inactive_ind, inv_type, shelf_
life, retest_interval, item_abccode, gl_class, inv_class, sales_class, ship_
class, frt_class, price_class, storage_class, purch_class, tax_class, customs_
class, alloc_class, planning_class, itemcost_class, cost_mthd_code, upc_code,
grade_ctl, status_ctl, qc_grade, lot_status, bulk_id, pkg_id, qcitem_no, qchold_
res_code, expaction_code, fill_qty, fill_um, expaction_interval, phantom_type,
whse_item_no, experimental_ind, exported_date, seq_dpnd_class, commodity_code,
ic_matr_days, ic_hold_days, attribute1, , attribute2, attribute3, attribute4,
attribute5, attribute6, attribute7, attribute8, attribute9, attribute10,
attribute11, attribute12, attribute13, attribute14, attribute15, attribute16,
attribute17, attribute18, attribute19, attribute20, attribute21, attribute22,
attribute23, attribute24, attribute25, attribute26, attribute27, attribute28,
attribute29, attribute30, attribute_category, user_name
```

Item Lot/Sublot Conversion API

```
item_no, lot_no, sublot_no, from_uom, to_uom, type_factor, user_name
```

Inventory Quantities API

```
trans_type, item_no, journal_no, from_whse_code, to_whse_code, item_um, item_
um2, lot_no, sublot_no, from_location, to_location, trans_qty, trans_qty2, qc_
grade, lot_status, co_code, orgn_code , trans_date, reason_code, user_name
```

Lot Create API

item_no, lot_no, subplot_no, lot_desc, qc_grade, expaction_code, expaction_date, lot_created, retest_date, strength, inactive_ind, origination_type, shipvendor_no, vendor_lot_no, ic_matr_date, ic_hold_date, attribute1, attribute2, attribute3, attribute4, attribute5, attribute6, attribute7, attribute8, attribute9, attribute10, attribute11, attribute12, attribute13, attribute14, attribute15, attribute16, attribute17, attribute18, attribute19, attribute20, attribute21, attribute22, attribute23, attribute24, attribute25, attribute26, attribute27, attribute28, attribute29, attribute30, attribute_category, user_name

Step 3 Log Files

When an API is run it generates a log file that describes what was generated during the API call or what went wrong. These log files must be placed in a specific directory on your system (determined by a parameter in your init.ora file) and that directory must have open permissions to write to that directory. To find out where this directory is, you should run the following query.

```
SQL> select value from v$parameter where name like 'utl%';
```

This query may return multiple directories, so you need to make sure that the directory you want your log files to write to exists, and that the permissions are open to write to it. This directory is entered in your wrapper file where you determine what you want the log file to be called. An example of the code is below. The directory that the log file is writing to in this example is /usr/tmp. The icqty.log is defined as the log file to be generated, and the other file listed is icqty.txt, the parameter file discussed in the next step.

```
GMI_QUANTITY_WRP.post ('/usr/tmp','icqty.txt','icqty.log','');
```

There is also a log file that is generated with the session number that can provide some additional information. That log file is system generated, and is named wrapperxxxxx.log, where the xxxxx is the session number. These two log files have similar information, but if you encounter problems this is another place to look for help.

Step 4 Running the API

The Inventory APIs are PL/SQL based and run database packages, and are run by SQL*Plus or Unix, initiating a sql session first.

Via Unix

```
sqlplus apps/apps@database @wrapper_file
```

Via SQL*Plus

Login into SQL*Plus, and start the wrapper file using @wrapper_file.

Index

A

ABC code, 4-4
action code, lot expiry, 4-6
adjust inventory, 1-6
alloc_class, 4-5
allocation class, 4-5
allowable deviation, factor, 4-3
alt_itema, 4-3
alt_itemb, 4-3
alternative name for item, 4-3
API, 1-1
 architecture, 1-8
API architecture, 1-8
API, definition, 1-2
application code calls, 1-8
application program interfaces, 1-2
architecture, layered, 1-8
ASCII flat file, 2-2, 2-5
asynchronous mode, 2-2
attribute_category, 4-8, 4-19
attributes, 4-6, 4-17
authorization, to a procedure, 2-3

B

Batch File, 1-4
bulk id, 4-5
bulk_id, 4-5

C

c_item_mst.grade_ctl, 4-16
calling program, 1-2

calling wrapper interrogation, 2-3
calls, from application code, 1-8
change lot status, 1-6
changing quality control grade, 1-6
character delimited, 2-5
co_code, 4-13
code calls, from the application, 1-8
code example, 2-6
Code Re-Use, 1-5
commit, 2-4
commodity code, 4-6
commodity_code, 4-6
company code, 4-13
Consistent Behavior, 1-5
conversion factor, 4-10
cost class, 4-5
cost method code, 4-5
cost_mthd_code, 4-5
CPG, 1-2
create inventory, 1-6
creating a lot, 1-6
creating an inventory item, 1-6
customs class, 4-5
customs_class, 4-5

D

data from legacy systems, 1-7
database tables, 2-3
default lot status, 4-5
default quality control grade, 4-5
definition, API, 1-2
dependencies and syntax, procedural, 2-3
descriptive flexfield segment, 4-6

deviation_hi, 4-3
deviation_lo, 4-3
dual unit of measure indicator, 4-3
dual UOM, 4-3
dualum_ind, 4-3

E

Ease of Integration, 1-5
ERP, 1-2
executing a stored procedure, 2-3
expaction_code, 4-6, 4-15
expaction_date, 4-15
expaction_interval, 4-6
experimental item indicator, 4-6
experimental_ind, 4-6
expire_date, 4-15, 4-16
expiry action code, 4-15
expiry action date, 4-15
expiry action interval, 4-6
export to Oracle financials date, 4-6
exported_date, 4-6

F

file type, nomenclature, 1-10
file type, .pls, 1-10
fill qty, 4-6
fill um, 4-6
fill_qty, 4-6
fill_um, 4-6
Financials (Oracle), date exported to, 4-6
flat file, 2-2
FND_API, 1-3
FND_API.G_FALSE, 3-6
FND_API.G_RET_STS_EXP_ERR, 4-2
FND_API.G_RET_STS_SUCCESS, 4-2
FND_API.G_RET_STS_UNEXP_ERR, 4-2
FND_MESSAGE, 1-3
FND_PUB_MSG, 1-3
freight class, 4-4
from location, 4-13
from unit of measure, 4-10
from warehouse code, 4-12
from_location, 4-13

from_uom, 4-10
from_whse_code, 4-12
frt_class, 4-4

G

GET MESSAGES, 1-5, 2-3
gl class, 4-4
gl_class, 4-4
GMF_AllocationDefinition_PUB, 1-10
GMF_BurdenDetails_PUB, 1-10
GMF_ItemCost_PUB, 1-10, 3-2, 3-3, 3-4, 3-5
GMF_ResourceCost_PUB, 1-10
GMFPALCS.pls, 1-10
GMFPBRDB.pls, 1-10
GMFPBRDS.pls, 1-10
GMFPCSTB.pls, 1-10, 3-2, 3-3, 3-4, 3-5
GMFPCSTS.pls, 1-10, 3-2, 3-3, 3-4, 3-5
GMFPRESS.pls, 1-10
GMI
 Default Lot, 4-13
 Intrastat, 4-6
GMIGAPI, 1-8
GMIGAPI.Create_Item_Lot_Conv, 4-9
GMIGAPI.Create_Lot, 4-14
GMIGUTL, 1-8
GMIGUTL.Get_Item, 4-10, 4-15
GMIGUTL.Get_Lot, 4-10
GMIPAPI, 1-8
GMIPAPI.Create_Item_Lot_Conv, 4-9
GMIPAPI.Create_Lot, 4-14
GMIPAPI.Inventory_Posting, 4-11, 4-12
GMIPILW, 1-9
GMIPITW, 1-9
GMIPLOW, 1-9
GMIPQTW, 1-9
GMIPTXN, 1-9
GMIPXFW, 1-9
GMIVBUL, 1-9
GMIVBUS, 1-9
GMIVCMP, 1-9
GMIVDBL, 1-9
GMIVILC, 1-9
GMIVITM, 1-9
GMIVLOC, 1-9

GMIVLOT, 1-9
GMIVPND, 1-9
GMIVQTY, 1-9
GMIVSUM, 1-9
GMIVTXN, 1-9
GMIVXFR, 1-9
grade_ctl, 4-5
group layer, 1-8

I

IC\$ALLOW_INACTIVE, 4-12
IC_API_INVALID_ABCCODE, A-4
IC_API_INVALID_ALLOC_CLASS, A-4
IC_API_INVALID_COMMODITY_CODE, A-5
IC_API_INVALID_CUSTOMS_CLASS, A-4
IC_API_INVALID_DEVIATION, A-3
IC_API_INVALID_DUALUM_IND, A-3
IC_API_INVALID_EXPACTION_CODE, A-5
IC_API_INVALID_EXPACTION_DATE_API, A-7
IC_API_INVALID_EXPERIMENTAL_IND, A-5
IC_API_INVALID_EXPIRE_DATE, A-7
IC_API_INVALID_FRT_CLASS, A-4
IC_API_INVALID_GL_CLASS, A-4
IC_API_INVALID_GRADE_CTL, A-4
IC_API_INVALID_HOLD_DAYS, A-5
IC_API_INVALID_INACTIVE_IND, A-4
IC_API_INVALID_INV_CLASS, A-4
IC_API_INVALID_INV_TYPE, A-4
IC_API_INVALID_ITEM, A-7
IC_API_INVALID_ITEM_NO, A-5, A-6
IC_API_INVALID_ITEMCOST_CLASS, A-4
IC_API_INVALID_JOURNAL_NO, A-6
IC_API_INVALID_LOCATION, A-6
IC_API_INVALID_LOCT_CTL, A-4
IC_API_INVALID_LOT_CTL, A-3
IC_API_INVALID_LOT_EXPACTION_CODE, A-7
IC_API_INVALID_LOT_HOLD_DAYS, A-8
IC_API_INVALID_LOT_INACTIVE_IND, A-7
IC_API_INVALID_LOT_INDIVISIBLE, A-3
IC_API_INVALID_LOT_MATR_DAYS, A-8
IC_API_INVALID_LOT_NO, A-5, A-6
IC_API_INVALID_LOT_ORIGINATION_
TYPE, A-7
IC_API_INVALID_LOT_QC_GRADE, A-7
IC_API_INVALID_LOT_SHIPVENDOR_NO, A-7
IC_API_INVALID_LOT_STATUS, A-6
IC_API_INVALID_LOT_STATUS_API, A-5
IC_API_INVALID_LOT_STRENGTH, A-7
IC_API_INVALID_LOT_UOM, A-5
IC_API_INVALID_LOT_UOM_TYPE, A-5
IC_API_INVALID_MATCH_TYPE, A-4
IC_API_INVALID_MATR_DAYS, A-5
IC_API_INVALID_NONINV_IND, A-4
IC_API_INVALID_PLANNING_CLASS, A-4
IC_API_INVALID_PRICE_CLASS, A-4
IC_API_INVALID_PURCH_CLASS, A-4
IC_API_INVALID_QC_GRADE, A-5, A-7
IC_API_INVALID_QCHOLD_RES_CODE, A-5
IC_API_INVALID_QCITEM_NO, A-5
IC_API_INVALID_QTY, A-6
IC_API_INVALID_RETEST_DATE, A-7
IC_API_INVALID_RETEST_INTERVAL, A-4
IC_API_INVALID_SALES_CLASS, A-4
IC_API_INVALID_SEQ_DPND_CLASS, A-5
IC_API_INVALID_SHIP_CLASS, A-4
IC_API_INVALID_STATUS_CTL, A-4
IC_API_INVALID_STORAGE_CLASS, A-4
IC_API_INVALID_SUBLOT_CTL, A-4
IC_API_INVALID_TAX_CLASS, A-4
IC_API_INVALID_TRANS_DATE, A-7
IC_API_INVALID_TRANS_TYPE, A-6
IC_API_INVALID_TYPE_FACTOR, A-5
IC_API_INVALID_UOM, A-3, A-6
IC_API_INVALID_WHSE_CODE, A-6
IC_API_INVALID_WHSE_ITEM_NO, A-5
IC_API_ITEM_ALREADY_EXISTS, A-3
IC_API_ITEM_CNV_ALREADY_EXISTS, A-5
IC_API_ITEM_LOT_UOM_FAILED, A-6
IC_API_ITEM_NOT_LOT_CTL, A-7
IC_API_LOCT_ONHAND_EXISTS, A-6
IC_API_LOT_ALREADY_EXISTS, A-7
IC_API_LOT_ITEM_UOM_MISMATCH, A-6
IC_API_MOVE_STATUS_ERR, A-6
IC_API_NEG_QTY_NOT_ALLOWED, A-6
IC_API_NO_LOCT_ONHAND, A-6
IC_API_NVALID_SHELF_LIFE, A-4
IC_API_SUBLOT_NOT_REQD, A-6, A-7
IC_API_TRANS_TYPE_FOR_ITEM, A-6
ic_hold_date, 4-17

- ic_hold_days, 4-6
- ic_item_cpg table, 4-2
- ic_item_mst, 4-15, 4-16
- ic_item_mst.expaction_code, 4-15
- ic_item_mst.expaction_interval, 4-16
- ic_item_mst.grade_ctl, 4-15
- ic_item_mst.ic_matr_days, 4-17
- ic_item_mst.lot_ctl, 4-13
- ic_item_mst.qc_grade, 4-15
- ic_item_mst.shelf_life, 4-16
- ic_item_mst.sublot_ctl, 4-13
- ic_jrnl_mst, 4-12
- ic_loct_inv, 1-9
- ic_lots_cpg, 4-14
- ic_lots_mst, 4-13, 4-15, 4-17
- ic_matr_date, 4-17
- ic_matr_days, 4-6
- ic_summ_inv, 1-9
- ic_tran_cmp, 1-9
- ic_tran_pnd, 1-9
- ic_whse_mst, 4-12, 4-13
- inactive indicator, 4-17
- inactive item indicator, 4-4
- inactive_ind, 4-4, 4-17
- indicator, lot indivisibility, 4-3
- Insulation from Changes, 1-5
- interrogation, by calling wrapper, 2-3
- intracompany transfer package, 1-9
- inv_class, 4-4
- inv_type, 4-4
- inventory calendar, 4-13
- inventory class, 4-4
- inventory item, creation, 1-6
- inventory item, lot creation, 1-6
- inventory quantities, 1-2
- inventory transaction packages, 1-9
- inventory type, 4-4
- Item Cost
 - Structure, 3-2, 3-3, 3-4, 3-5
- item creation, 1-2, 1-6
- item description 1, 4-3
- item description 2, 4-3
- Item Lot/Sublot Conversion, 1-2
- item lot/sublot unit of measure conversion, 1-6
- item number, 4-2, 4-10, 4-12

- item_abccode, 4-4
- item_desc1, 4-3
- item_desc2, 4-3
- item_no, 4-2, 4-10, 4-12
- item_um, 4-3, 4-13
- item_um2, 4-3, 4-13
- itemcost_class, 4-5

J

- journal number, 4-12
- journal_no, 4-12

L

- layer
 - database access, 1-8
 - group, 1-8
 - private, 1-8
 - public, 1-8
 - validation, 1-8
 - wrapper, 1-8
- layered architecture, 1-8
- level code, 4-3
- level_code, 4-3
- location control, 4-4
- loct_ctl, 4-4
- logical transaction, 2-4
- lot controlled item indicator, 4-3
- Lot Create, 1-2
- Lot Create stored procedure, 3-5
- lot created date, 4-16
- lot creation, 1-6
- lot description, 4-15
- lot expiration, action code for, 4-6
- lot expire date, 4-16
- lot hold days, 4-6
- lot indivisibility, 4-3
- lot maturity days, 4-6
- lot number, 4-10, 4-13, 4-15
- lot status, 4-13
- lot status controlled item indicator, 4-5
- lot_created, 4-15, 4-16, 4-17
- lot_ctl, 4-3
- lot_desc, 4-15

lot_indivisible, 4-3
lot_no, 4-10, 4-13, 4-15
lot_status, 4-5, 4-13

M

match type, 4-4
match_type, 4-4
message count, 4-2
message encoding, 3-7
message list initialization, 3-6
message list, specifying number of messages
 added, 3-7
message stack, 4-2
move inventory, 1-6

N

network traffic, 2-3
noninv_ind, 4-4
non-inventory item, 4-4

O

Online User Interface (UI), 1-4
Oracle Financials, 4-6
Oracle Messages, 1-5
Oracle Messages table, 2-3
Oracle-supplied wrapper code, 1-8
organization code, 4-13
orgn_code, 4-13
origination type, 4-17
origination_type, 4-17

P

p_api_version, 3-6
p_commit, 3-7
p_ic_item_mst_cpg, 4-15
p_ic_item_mst_row, 4-10, 4-15
p_ic_lots_mst_row, 4-10
p_init_msg_list, 3-6, 4-15
p_item_rec, 4-2
package body, 1-8
package body, nomenclature, 1-10

package specification, 1-8
packages, inventory transactions, 1-9
parameters, 1-2
permissions, 2-3
phantom type, 4-6
phantom_type, 4-6
pkg id, 4-5
pkg_id, 4-5
planning class, 4-5
planning_class, 4-5
platform independence, 2-3
.pls file type, 1-10
PL/SQL, 1-2, 2-2, 2-3
 record, 2-3
po_vend_mst, 4-17
price class, 4-4
price_class, 4-4
primary transaction quantity, 4-13
primary unit of measure, 4-13
private layer, 1-8
private package, 1-8
procedure, 1-8
procedure execution, 2-3
procedure, stored, 2-3
procedures stored, 2-2
processing standard message functions, 3-7
public layer, 1-8
public package, 1-8
purch_class, 4-5
purchase class, 4-5

Q

qc_grad_mst, 4-13, 4-15
qc_grade, 4-5, 4-13, 4-15
qchold_res_code, 4-6
qcitem_no, 4-6
quality control
 grade, 4-13
 grade controlled item indicator, 4-5
 hold reason code, 4-6
 reference item, 4-6

R

reason code, 4-13
reason_code, 4-13
record validation, 2-3
record, PL/SQL, 2-3
retest date, 4-16
retest interval, 4-4
retest_date, 4-16
retest_interval, 4-4
Robust Validation, 1-6

S

sales class, 4-4
sales_class, 4-4
second alternative name for item, 4-3
secondary transaction quantity, 4-13
secondary unit of measure, 4-3, 4-13
security, 2-3
separation, relating to user interface, 2-3
seq_dpnd_class, 4-6
sequence dependent class, 4-6
shelf life, 4-4
shelf_life, 4-4
ship class, 4-4
ship_class, 4-4
shipvendor_no, 4-17
specification, nomenclature, 1-10
standard message function processing, 3-7
status_ctl, 4-5
storage class, 4-5
storage_class, 4-5
stored procedure, 2-2, 2-3
 body, 2-2
 execution, 2-3
 specification, 2-2
strength, 4-17
subplot control, 4-3
subplot number, 4-10, 4-13, 4-15
subplot_ctl, 4-3, 4-15
subplot_no, 4-10, 4-13, 4-15
support policy, 1-3
SY\$CPG_INSTALL, 4-2, 4-14, 4-15
SY\$MAX_DATE, 4-17

SY_API_INVALID_CO_CODE, A-7
SY_API_INVALID_OP_CODE, A-7
SY_API_INVALID_ORGN_CODE, A-7
SY_API_INVALID_REASON_CODE, A-7
SY_API_UNABLE_TO_GET_DOC_NO, A-6
SY_API_UNABLE_TO_GET_SURROGATE, A-5,
 A-7, A-8
sy_orgn_mst, 4-13
sy_orgn_mst.co_code, 4-13
sy_reas_cds, 4-13
sy_type_mst, 4-17
sy_uoms_mst, 4-13
synchronous mode, 2-2

T

table, Oracle Messages, 2-3
table_name, 4-17
tax class, 4-5
tax_class, 4-5
Technical Overview
 Item Cost, 2-1
temporary table, 2-2
third party code, 1-8
to location, 4-13
to unit of measure, 4-10
to warehouse code, 4-13
to_location, 4-13
to_uom, 4-10
to_whse_code, 4-13
traffic, network, 2-3
trans_date, 4-13
trans_qty, 4-13
trans_qty2, 4-13
trans_type, 4-12, 4-13
transaction date, 4-13
transaction packages, inventory, 1-9
transaction type, 4-12
transaction, logical, 2-4
type_factor, 4-10

U

unit of measure, 4-3
upc_code, 4-5

user interface layer, 2-3
user name, 4-6, 4-10, 4-13
user_name, 4-6, 4-10, 4-13
UTL_FILE.INVALID_OPERATION, A-8
UTL_FILE.INTERNAL_ERROR, A-8
UTL_FILE.INVALID_FILEHANDLE, A-8
UTL_FILE.INVALID_MODE, A-8
UTL_FILE.INVALID_PATH, A-8
UTL_FILE.READ_ERROR, A-8
UTL_FILE.WRITE_ERROR, A-8

x_ic_jrnl_mst_row, 4-11
x_ic_lots_cpg, 4-14
x_ic_lots_cpg_row, 4-14
x_ic_lots_mst_row, 4-14
x_msg_count, 3-7
x_msg_data, 3-7
x_return_status, 3-7

V

validating a record, 2-3
Value-ID Conversion, 3-7
variables, 1-8
vendor_lot_no, 4-17
version compatibility, validation, 3-6

W

warehouse item number, 4-6
whse_item_no, 4-6
wrapper, 1-9
 GMIPILW, 1-9
 GMIPITW, 1-9
 GMIPLOW, 1-9
 GMIPQTW, 1-9
 item creation, 1-9
 item/lot UOM conversion, 1-9
 lot creation, 1-9
 quantity transactions, 1-9
wrapper code, 1-8
wrapper function, 2-2, 2-5, 3-2, 3-3, 3-4
wrappers, 1-8

X

x_ic_adjst_jnl_row, 4-11
x_ic_adjst_jnl_row1, 4-11
x_ic_adjst_jnl_row2, 4-11
x_ic_item_cnv_row, 4-9
x_ic_item_cpg_row, 4-2
x_ic_item_cpg_row, 4-2
x_ic_item_mst_row, 4-2

