Oracle® Process Manufacturing

Inventory APIs User's Guide

Release 11*i*

October 2000

Part No. A82921-02

ORACLE

Oracle® Process Manufacturing Inventory APIs User's Guide, Release 11*i*

Part No. A82921-02

Primary Author: Richard D. Persen

Contributing Authors: Phil Mellows, Paul Schofield, Harminder Verdding

Contents

2 Item Create API

3 Item Create API Wrapper

4 Item Lot/Sublot Conversion API

5 Item Lot/Sublot Conversion API Wrapper

6 Inventory Quantities API

7 Inventory Quantities API Wrapper

8 Lot Create API

9 Lot Create API Wrapper

Index

Send Us Your Comments

Oracle® Process Manufacturing Inventory APIs User's Guide, Release 11*i*

Part No. A82921-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

If you find any errors or have any other suggestions for improvement, please indicate 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, and telephone number below.

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

Preface

Welcome to the Oracle Process Manufacturing *Inventory APIs User's Guide*. This user's guide includes the information you need to work with the Oracle Process Manufacturing (OPM) application effectively.

This preface explains how this user's guide is organized and introduces other sources of information that can help you.

Intended Audience

This guide assumes that you have working knowledge of your business area's processes and tools. It also assumes that you are familiar with OPM Inventory APIs. If you have never used OPM Inventory APIs, we suggest you attend one or more of the Oracle Process Manufacturing training classes available through Oracle World Wide Education.

This guide also assumes that you are familiar with the Oracle Applications graphical user interface. To learn more about Oracle Applications graphical user interface, read the *Oracle Applications User's Guide*.

About This Guide

This guide contains overviews as well as task and reference information. It includes the following:

Name	Description
OPM Inventory APIs - Introduction	Presents Inventory API technical requirements, technical structure, architecture, and message handling recommendations. This topic also covers Inventory API installation and upgrade issues, and provides engineering notes on this release.
Item Create API	Provides the business function and technical details for the Item Create API
Item Create API Wrapper	Provides the business function and technical details for the Item Create API wrapper
Item Lot/Sublot Conversion API	Provides the business function and technical details for the Item Lot/Sublot conversion API
Item Lot/Sublot Conversion API Wrapper	Provides the business function and technical details for the Item Lot/Sublot conversion API wrapper
Inventory Quantities API	Provides the business function and technical details for the Inventory Quantities API
Inventory Quantities API Wrapper	Provides the business function and technical details for the Inventory Quantities API wrapper
Lot Create API	Provides the business function and technical details for the Lot Create API
Lot Create API Wrapper	Provides the business function and technical details for the Lot Create API wrapper

Information Sources

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

Online Documentation

Oracle Applications documentation is available on CD-ROM, except for technical reference manuals. User's guides are available in HTML format and on paper. Technical reference manuals are available on paper only. Other documentation is available on paper and sometimes in PDF format.

The content of the documentation remains the same from format to format. Slight formatting differences could occur due to publication standards, but such differences do not affect content. For example, page numbers are included on paper, but are not included in HTML.

The HTML documentation is available from all Oracle Applications windows. Each window is programmed to start your web browser and open a specific, context-sensitive section. Once any section of the HTML documentation is open, you can navigate freely throughout all Oracle Applications documentation.

Related Documents

Oracle Process Manufacturing shares business and setup information with other Oracle products. You may find the following Oracle Applications user's guides useful:

- *Oracle Applications User's Guide*
- *Oracle Application's Flexfields Guide*
- *Oracle Workflow User Guide*
- *Oracle Applications System Administrator's Guide*
- *Oracle General Ledger User's Guide*
- *Oracle Payables User's Guide*
- *Oracle Receivables User's Guide*
- *Oracle Human Resources North American User's Guide*
- *Oracle Purchasing User's Guide*

Oracle Process Manufacturing Guides

The following is a list of documentation in each product group for OPM:

Financials

- *Oracle Process Manufacturing Accounting Setup User's Guide*
- *Oracle Process Manufacturing Cost Management User's Guide*

- *Oracle Process Manufacturing Manufacturing Accounting Controller User's Guide*
- *Oracle Process Manufacturing and Oracle Financials Integration User's Guide*

Inventory Control

- *Oracle Process Manufacturing Intrastat Reporting User's Guide*
- *Oracle Process Manufacturing Inventory Management User's Guide*
- *Oracle Process Manufacturing Physical Inventory User's Guide*

Logistics

- *Oracle Process Manufacturing Order Fulfillment User's Guide*
- *Oracle Process Manufacturing Purchase Management User's Guide*

Process Execution

- *Oracle Process Manufacturing Process Operation Control User's Guide*
- *Oracle Process Manufacturing Production Management User's Guide*

Process Planning

- *Oracle Process Manufacturing Capacity Planning User's Guide*
- *Oracle Process Manufacturing Integration with Advanced Planning and Scheduling User's Guide*
- *Oracle Process Manufacturing MPS/MRP and Forecasting User's Guide*

Product Development

- *Oracle Process Manufacturing Formula Management User's Guide*
- *Oracle Process Manufacturing Laboratory Management User's Guide*
- *Oracle Process Manufacturing Quality Management User's Guide*

Regulatory

- *Oracle Process Manufacturing Regulatory Management User's Guide*

System Administration and Technical Reference

- *Oracle Process Manufacturing Implementation Guide*
- *Oracle Process Manufacturing System Administration User's Guide*
- Oracle Process Manufacturing Technical Reference Manuals

Training

Oracle offers a complete set of formal training courses to help you master Oracle Process Manufacturing and reach full productivity quickly. We organize these courses into functional learning paths, so you take only those courses appropriate to your area of responsibility.

You have a choice of educational environments. You can attend courses offered by Oracle Education Services at any one of our many Education Centers, or you can arrange for our trainers to teach at your facility. 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's structure, terminology, and data as examples in a customized training session delivered at your own facility.

Conventions

The following conventions are used in this guide:

Bolded Text

Buttons, fields, keys, menus, and selections are bolded in procedures only. For example: To access the next window, click **OK**. Otherwise, references to these features appear in regular type.

Additional Menu Options

Only nonstandard menu options are discussed. Standard menu bar options (such as Save) are not discussed. These standard options are described in the *Oracle Applications User's Guide*. Only menu options unique to the use of the specific window are discussed.

Field References

References to fields within procedures are in bold type. References within the body of this guide appear in regular type.

Required Fields

The word Required appears as the last word in the field description of all required fields. When the field is required contingent on the entry in another field, or only in specific situations, "Required if..." is the last sentence of the field description.

Fields Reserved for Future Use

Fields with no current processing implications are referenced by the statement "This field is not currently used" or "Reserved for future use." Do not use these fields for

your own reference data, because there are plans to link future functionality to these fields. Fields intended for informational purposes only are referenced by the statement "This field is for informational purposes only."

Pending/Completed Transactions

Discussions about processing transactions that use the words pending and completed refer to the status of a transaction. Pending and completed do not refer to the database tables that are updated as a result of transactions (for example, some completed transactions are stored in the Pending Transactions table).

Procedures

Most topics contain a procedure with numbered steps. Any actions which are subordinate to a step are assigned letters. You can customize your Oracle Application, therefore, all procedures are suggestive only. Navigate to windows and between responsibilities in a way that works best for your particular setup. Also note that fields may appear in a different order than they are discussed.

Use of the Word Character

The word character means an alphanumeric character. Characters that are numeric or alphabetic only are referenced specifically. Depending on your system security profile, you many not have access to all of the windows and functions described in this guide. If you do not see a menu option described in this guide, and you want access to it, contact your System Administrator.

Do Not Use Database Tools to Modify Oracle Applications Data

Oracle Applications tables are interrelated. As a result, any change you make using Oracle Applications can update many tables at once. If you modify the Oracle Applications data using anything other than Oracle Applications, you could change a row in one table without making corresponding changes in related tables. If your tables are not synchronized with each other, you risk retrieving erroneous information and receiving 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 track who changes information. If you enter information into database tables using database tools, you could 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.

Consequently, we strongly recommend that you never use SQL*Plus or any other tool to modify Oracle Applications data unless otherwise instructed by Oracle Support Services.

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 45 software modules for financial management, supply chain management, manufacturing, project systems, human resources, sales, and service 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 140 countries around the world.

Thank You

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

We value your comments and feedback. At the beginning of this guide is a Reader's Comment Form that you can use to explain what you like or dislike about 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 an electronic mail message to appsdoc@us.oracle.com

1

OPM Inventory APIs - Introduction

This topic provides a basic understanding of Oracle Process Manufacturing (OPM) Inventory APIs. It presents the technical requirements for stored procedures. You will gain an understanding of the technical structure and architecture of the Inventory APIs. Also included in this topic is a technical overview of Inventory APIs as well as how to handle error messages and result messages. Installation and upgrade recommendations are provided with engineering notes on this release.

The following topics are covered:

- Understanding OPM Inventory APIs
- Stored Procedures Technical Requirements
- Inventory APIs - Technical Structure and Architecture
- Technical Overview of Inventory APIs
- Error Message Handling
- Result Message Handling
- Installation and Upgrade
- Engineering Notes on This Release

Understanding OPM Inventory APIs

Oracle Process Manufacturing (OPM) is a key component to Enterprise Resource Planning (ERP) solutions. For example, it is a key component to Consumer Packaged Goods (CPG), 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 (APIs), OPM is able to integrate with touch points either as or within an ERP solution. APIs are a documented, supported method for communicating within or between modules. The APIs support external interfaces to OPM.

The APIs discussed in this document support the OPM Inventory Control module. These interfaces make use of the standard functionality and logic implemented in that module. The APIs provide hooks into which customers can interface their own programs. This guide describes how the stored procedures should be called, the parameters that are required (and those that are optional) and the values that are returned to the calling program. This includes all error conditions that may arise.

The APIs and related functions discussed in this document are:

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

Stored Procedures Technical Requirements

The OPM Inventory API stored procedures are designed to operate in an OPM 11i environment only.

The procedures make use of the following standard Oracle Applications packages to perform message handling:

- 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. The application messages are stored in the database table fnd_new_messages.
- FND_PUB_MSG - the standard Oracle Applications message retrieval function, used to interrogate the procedure messages.

These packages are created and owned by the applications user as part of a standard Oracle Applications installation. They are not supplied within the OPM Inventory APIs' installation.

You should become familiar with the functionality of the procedures and functions contained in the FND packages. The calling program utilizes these procedures on return from the API in order to decode and handle messages.

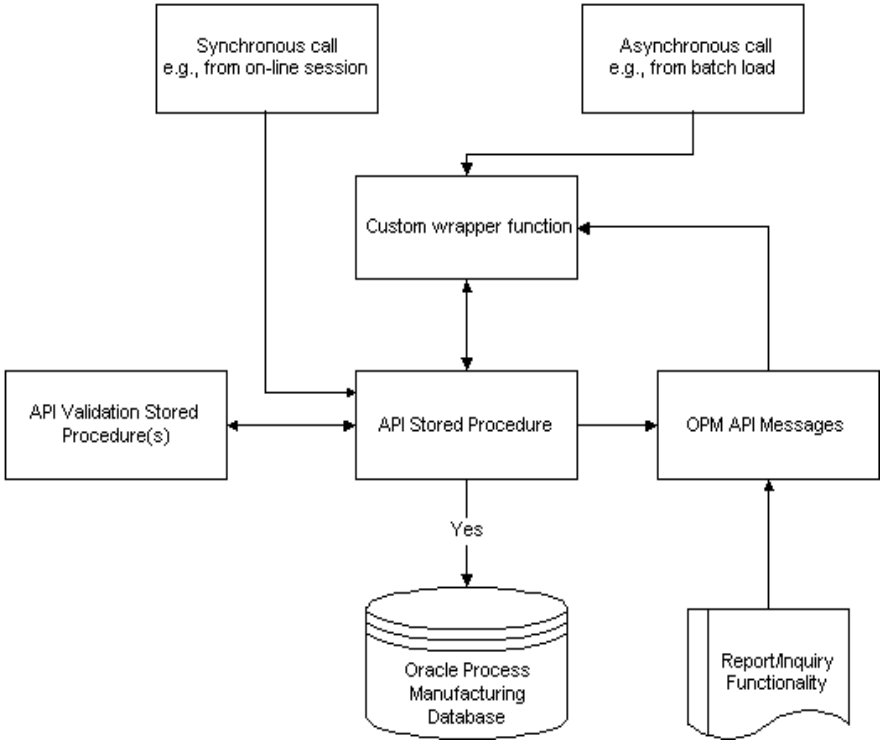
Inventory APIs - Technical Structure and Architecture

The following topic covers the:

- Stored Procedure Mechanism
- API Architecture
- API Package Details
- Item Master Block Relationship Diagram

Stored Procedure Mechanism

The following is a schematic of the Oracle Process Manufacturing (OPM) stored procedure API mechanism. It shows the structure of the API and the mechanism needed to execute it. It also indicates the additional tables and functionality required to both execute and audit results.

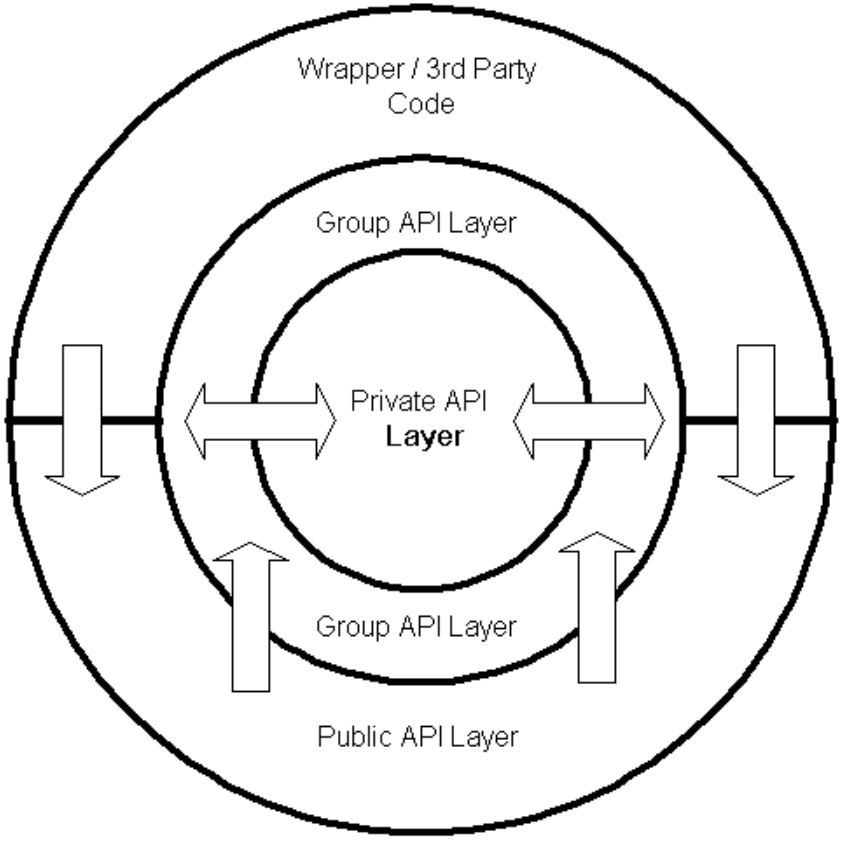


API Architecture

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 - within which there are validation and database access layers

This structure is illustrated in the following diagram. Arrows indicate permitted calls between layers.



API Package Details

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

The APIs use a layered architecture, as described in the API Architecture topic. Third party and Oracle-supplied wrapper code should only access the functions, procedures and variables that are contained in the public / group package specifications. Private packages should only be accessed from the group code and from nowhere else.

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

- P (for public)
- V (for private)
- G (for 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 and so forth.
- GMIGAPI - for the group level API routines. These are called by the routines in the GMIPAPI package and they are also 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 should be 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 should be 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 can be 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:

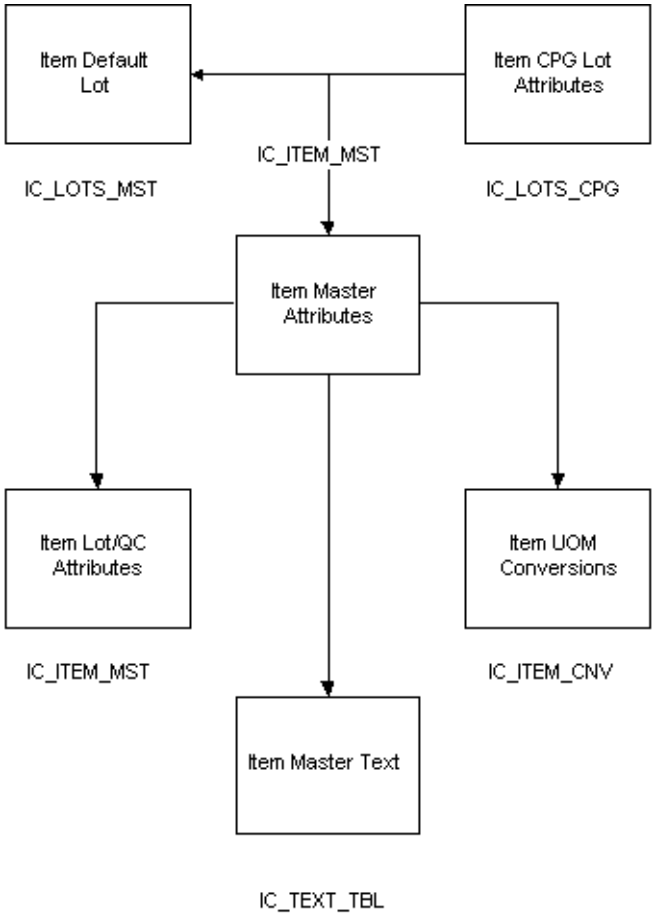
- GMIPITW - Wrapper for item creation
- GMIPLOW - Wrapper for lot creation
- GMIPILW - Wrapper for item/lot UOM conversion creation
- GMIPQTW - Wrapper for quantity transactions

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

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 lower case letters.

Item Master Block Relationship Diagram

The logical view of the Item Master is represented in the following graphic. This suggests the business characteristics of the Item Master, not its database design.



The Item Master table IC_ITEM_MST holds all attributes of the item.

Lot controlled and quality control attributes are prompted for in the maintenance form once the appropriate indicators are set. These details are also held against the item.

Whether an item is defined as lot controlled or not lot controlled, the system always generates a DEFAULT lot against the item on IC_LOTS_MST.

All lots hold additional attributes for Consumer Packaged Goods (CPG) specific processing. These are held against IC_LOTS_CPG.

When an item is identified as dual UOM and the alternate UOM is of a different type (for example, item is defined in type WEIGHT with dual UOM in VOLUME) the save of the item master prompts for the conversion factor between WEIGHT and VOLUME for the item. This mechanism and structure supports transactional processing of the item in any supported UOM. This is handled outside of the Item Create procedure through an additional function (Insert_Ic_Item_Cnv).

Technical Overview of Inventory APIs

This section describes all elements in the structure. The stored procedures may be called from a user wrapper function which executes the procedures and deal with the return status and messages from the execution.

API - Input Data Sources

The following are API Data Sources:

Flat File/Batch File - Asynchronous Mode

Input data to the user wrapper function may come from a flat file source. This may be processed by the wrapper and the details passed, as parameters, to the stored procedure in asynchronous mode. In this mode if a single record is processed then the commit flag can be passed to the API so that the API commits if successful. If multiple records are 'batched' for processing then the API is called with the commit flag not set so the API has no responsibility for committing. In this case the calling function monitors the success or failure (return code) from the called procedure and COMMIT based on that result.

Online User Interface (UI) - Synchronous Mode

Input data to the API may come directly from a form or other user interface. This may be processed by the UI and the details passed, as parameters, to the stored procedure in synchronous mode. In this mode, the UI calling function monitors the success or failure (return code) from the called procedure and COMMIT based on that result.

Wrapper Function

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 messages:

- Successful
- Error (Unsuccessful)
- Unexpected error

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

Wrapper Function - Input Data Sources

The following are input data sources for the wrapper function.

Flat File

Input data to the user wrapper function may come from a flat file source. The flat file is read record by record. For each record the wrapper builds the necessary parameters and call the Item Create API. The wrapper assumes no inter-dependency between the records and therefore instructs the API to commit each successfully processed record.

Temporary Table

Input data to the user wrapper function may 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 Item Create 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 compiled version of 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 allows a global level of authorization against the procedure rather than a table by table grant discipline within an application which could hold anomalies. If you have grant authority to the procedure then it should be assumed you have permissions at the database, and permissions at the 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

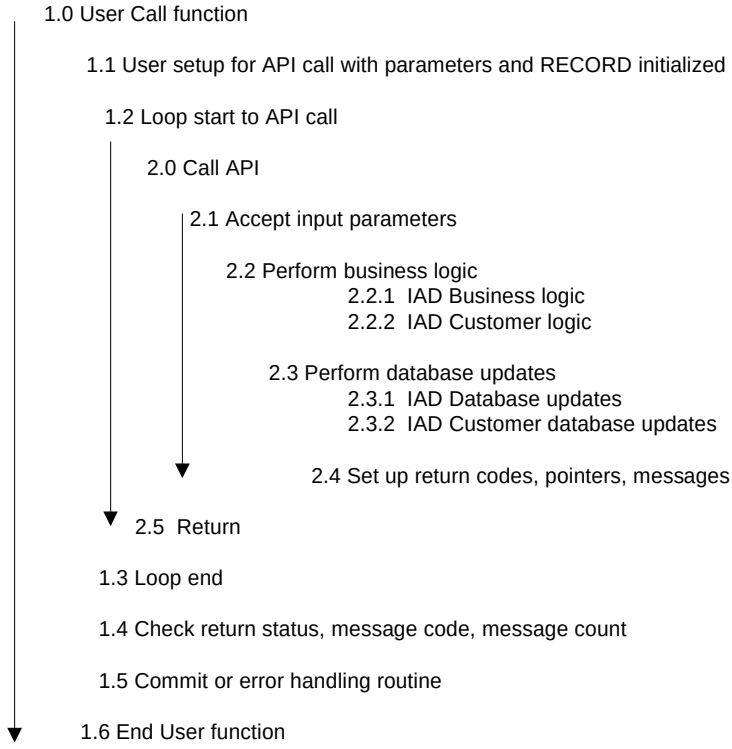
The stored procedure may be called with the appropriate parameters forwarded in a PL/SQL RECORD format as detailed. The procedure validates the RECORD and then processes the appropriate functional logic as required. The procedure writes appropriate messages to the Oracle Messages table. These are informational (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 may call other validation procedures in the course of its execution - example 'validate UOM code'.

Note: Functions called by these procedures do not use IN OUT parameters. This facility was disapproved from Oracle 8 onward.

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

The following is an outline of the stored procedure execution.



The Item Master table IC_ITEM_MST holds all attributes of the item.

Lot controlled and QC attributes are prompted for in the maintenance form once the appropriate indicators are set. These details are also held against the item.

Whether an item is defined as lot controlled or not the system always generates a DEFAULT lot against the item on IC_LOTS_MST.

All lots hold additional attributes for Consumer Packaged Goods (CPG)-specific processing. These are held against IC_LOTS_CPG.

Common Stored Procedure Parameters

All stored procedure APIs are called with PL/SQL parameters. Some of these parameters are common to all API activities while others are dependent on the activity. The following PL/SQL parameters are common to all API activities:

Parameter	Type	IN/OUT	Required	Validation
p_api_version	number	IN	Y	Used for version compatibility. The version sent by the calling function is compared to the internal version of the API and an unexpected error is generated if they do not match.
p_init_msg_list	varchar2	IN	N	Used to specify whether the message list should be initialized on entry to the API. It is an optional parameter, and if not supplied, defaults to FND_ API.G_FALSE which means that the API does not initialize the message list.
p_commit	varchar2	IN	N	Used to specify whether the API should commit its work before returning to the calling function. If not supplied it defaults to FND_API.G_FALSE.
p_validation_level	varchar2	IN	N	Used internally. Application code should not specify anything other than the default.
x_return_status	varchar2	OUT		Specifies whether the API was successful or failed: '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		This parameter returns the messages in encoded format. These messages can then be processed by the standard message.

Examination of both the x_return_status and x_msg_count indicates the pass or fail status of the call.

Error Message Handling

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

- 'S' - Successful
- 'E' - Expected error
- 'U' - Unexpected error

Error Messages for the individual APIs and wrappers are listed at the end of their respective topics.

Result Message Handling

APIs put result messages into a message list. Programs calling APIs can then get the messages from the list and process them appropriately. This may done by calling the API from an interactive process, or by writing messages to database tables or log files when calling the API from a batch process.

Messages are stored in an encode format to enable API callers to find out message names by using the standard functions provided by the message dictionary.

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 identified in the list that follows.

The following utility functions are defined in the FND_MSG_PUB package, in the file AFASMSGS.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, 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 get 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 five 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

Installation and Upgrade

Use the install scripts provided with the OPM patch.

Engineering Notes On This Release

In addition to resolving outstanding bugs in the previous releases of the Inventory APIs, the following issued have been addressed:

Public Code Not Accessed from the Private Layer

While the architecture described in the API Architecture topic was for the most part the structure used in earlier releases of this API, performance was compromised by calling public procedures from the private layer. In several cases this resulted in the same data being selected from the database several times. With this release, public code is never accessed from the private layer.

Single Retrieval of Profile Options

Repeated retrieval of profile options was identified as a major bottleneck in the API code. Each procedure always read the options it needed. In this release, all options are read once in a new setup procedure, which takes a user name as its parameter. All rows written to the database are tagged with this identity. The user names within the record structures which are passed to the APIs have been retained to minimize disruption to existing code, however these are ignored.

Reduced Data Validation

Other improvements, in line with performance analysis, include a reduction in the amount of validation performed; data that is not needed for a particular API is now ignored rather than reported as an error.

Reduced Database Access Within Validation Layer

Database access within the validation layer has been reduced. An example is that when a lot is created, the validation used to check that the same lot did not already exist. Customer input has indicated that errors are the exception rather than the norm, therefore, this check has now been postponed. If the lot exists, the error is trapped when the insertion fails, saving a select per API call.

In keeping with the strategy of minimizing database accesses, the functionality of the APIs now mirrors more closely the on-line functionality. For example when a lot is created on-line it inherits the quality control Grade from the item master. The previous version of the APIs took the value from the input parameters, after checking that it was valid, which required a database access. This defaulting of data has been adopted in several places, but the ability to override the default has been retained. If the input value is specified as NULL, it is defaulted whenever possible. If a value is specified, it is validated with the consequent overhead.

Error Reporting Broadened to Entire Input Record

Error reporting is now carried out on an entire input record rather than terminating at the first error located. However, where it would be useless to continue error checking, the previous behavior has been retained. Some error messages now cover several possible errors:

- In item creation, the message 'One or more classes is/are invalid' is self explanatory. It arises from the method used to validate all the classes in a single database hit, with a consequent positive effect on performance, rather than individually.
- The message 'Invalid Organization' in the quantities transaction validation means:
  - that the organization does not belong to the company, or
  - the organization's company is not the ultimate owner of the warehouse
  - that one or both of the warehouse codes does not exist, or
  - the company does not exist, or
  - that the organization does not exist.

  This arises because all of these conditions are checked in a *single* database retrieval, and it is therefore impossible to know which particular condition has not been met.

Checks Account for Nulls

Checks on return values now take account of nulls. In a few places vestigial code from the GEMMS/OPM 4.1x implementation was still checking for spaces or zero, which meant that some parts of the code could never be executed.

Intracompany Transfers Supported

New packages to support intracompany transfers have been included, although in this first release of the package there is no functionality behind the procedures.

PL/SQL Manages Inventory Transactions

A PL/SQL engine manages inventory transactions. This also contains public calls for use by third party code if this is ever needed.

Procedural Interfaces Modified

Most procedural interfaces have changed in line with customer comments. For example, when the create item API is called, the API now returns the database rows created as PL/SQL records so that the calling code has immediate knowledge of the surrogate keys that have been assigned. Previously an extra database call was needed to find each of these.

Utility Routines Called Internally by API Code

Several utility routines are called internally by the API code. For convenience, these are also callable by third party code. For the complete list with exact specifications, refer to the GMIGUTLS.pls file:

- GMIGUTL.Get_Item - reads in item rows according to the key or keyspecified
- GMIGUTL.Get_Lot - reads in lot rows according to the key or keys specified
- GMIGUTL.Get Reason - reads in the reason code for the key specified
- GMIGUTL.Setup - sets up various global constants, and so forth.

Global Variables and Flags Control API Behavior

Also residing in GMIGUTL are several global variables and flags which control API behavior. These are:

- IC$DEFAULT_LOT - name assigned to the default lot
- IC$DEFAULT_LOCT - name of the default location
- IC$ALLOWNEGINV - flag used to decide if inventory can be driven negative
- IC$MOVEDIFFSTAT - flag that controls how status of lots is transferred in movements
- IC$API_ALLOW_INACTIVE - flag used to allow or block transactions against inactive items and/or lots
- SY$INTRASTAT - flag that controls commodity code validation in the Create_ Item API
- SY$CPG_INSTALL - flag that determines whether the CPG tables, ic_lots_cpg and ic_item_cpg, are read and written to by the APIs
- DEFAULT_USER_ID - the internal ID for the user name specified as a parameter to GMIGUTL.Setup
- API_VERSION - a constant that is used to ensure compatibility in the API calls

- DEFAULT_LOGIN - the process ID for the current session

All of the externally callable API routines make calls to routines in the private layer, either directly or indirectly. If the call succeeds, other routines in the database layer are called to create or retrieve the rows. The routines in the private layer are not intended to be called by anything other than the routines in the group layer and are not supported outside this context.

2

Item Create API

This topic provides an explanation of the business function and technical overview of the Item Create API. The topic presents parameters used and includes table and view usage, package and procedure names, and special logic used in the Item Create API. Common error messages are listed.

The following topics are covered:

- Item Create API - Business Function
- Item Create API - Technical Overview
- Item Create API - Parameters and Interface
- Item Create API - Table and View Usage
- Item Create API - Package / Procedure Names
- Item Create API - Special Logic
- Item Create API - Error Messages

Item Create API - Business Function

This stored procedure is concerned with the following function within the Oracle Process Manufacturing (OPM) Inventory Management Module:

- Creating an inventory item

This topic describes how the stored procedure should be called, mandatory or optional parameter requirements and the values that are returned to the calling program. This includes all error conditions that may arise.

The procedure is intended as a create function only, used primarily to load item data from legacy systems on implementation. The API does not allow Item Update or Unit of Measure Conversion maintenance. These are handled through additional stored procedures such as discussed in the Item Lot/Sublot Conversion API - Special Logic topic.

Item Create API - Technical Overview

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 which is detailed in this topic. 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.

Item Create 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
);
```

The first 4, and last 3 parameters are standard across all of the API calls.

Parameter	Validation
p_api_version	Used internally to check that the call is compatible with user code. The latest patch documentation should be consulted for any possible changes.
p_init_msg_list	Determines if the error message stack should be purged. Usually this parameter is omitted and defaulted as above
p_commit	Determines whether the API should commit the new item after creation (assuming the creation succeeded). Normally this parameter is omitted and defaulted as above
p_validation_level	Determines whether validation is performed on the input record. Normally this parameter is omitted and defaulted as above. When an API is called from third party code no value other than the default is supported. In some internal calls the APIs call each other with other values to bypass sections of validation code.
p_item_rec	Contains the raw data to construct the item

If the creation was successful the x_ic_item_mst_row and x_ic_item_cpg_row parameters are returned with the data set up in the 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. Please refer to the Item Create API Wrapper topic for an example of how to populate this parameter and call the stored procedure.

Field/Column	Type	Length	Default	Req'd	Validation
item_no	varchar2	32			Duplicates not allowed on ic_ item_mst
item_desc1	varchar2	70		Y	Non-blank
item_desc2	varchar2	70		N	
alt_itema	varchar2	32		N	
alt_itemb	varchar2	32		N	
item_um	varchar2	4		Y	Must exist on sy_uoms_mst
dualum_ind	number	5	0	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	varchar2	4		N	Mandatory if dual UOM > 0. If non-blank then must exist on sy_uoms_ mst

Field/Column	Type	Length	Default	Req'd	Validation
deviation_lo	number		0	N	Must not be negative value. Must be 0 if dualum_ind = 0 or 1
deviation_hi	number		0	N	Must not be negative value. Must be 0 if dualum_ind = 0 or 1
level_code	number	5	0	N	Not currently used
lot_ctl	number	5		Y	0 = Not lot controlled 1 = Lot controlled
lot_indivisible	number	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	number	5	0	N	0 = Not sub lot controlled 1 = Sub lot controlled Must be 0 if lot_ctl = 0

Field/Column	Type	Length	Default	Req'd	Validation
loct_ctl	number	5	0	N	0 = Not location controlled 1 = Location controlled with validation of location 2 = Location controlled with no validation of location
noninv_ind	number	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	number	5	1	N	Type of invoice matching done Blank is no matching 1 = Invoice only 2 = Two way matching 3 = Three way matching
inactive_ind	number	5	0	N	0 = Active 1 = Inactive
inv_type	varchar2	4		N	Must exist on ic_invn_typ if non-blank

Field/Column	Type	Length	Default	Req'd	Validation
shelf_life	number		0	N	Must not be negative. Must be 0 if grade_ctl = 0
retest_interval	number		0	N	Must not be negative. Must be 0 if grade_ctl = 0
item_abccode	varchar2	4		N	
gl_class	varchar2	8		N	Must exist on ic_gled_cls if supplied
inv_class	varchar2	8		N	Must exist on ic_invn_cls if supplied
sales_class	varchar2	8		N	Must exist on ic_sale_cls if supplied
ship_class	varchar2	8		N	Must exist on ic_ship_cls if supplied
frt_class	varchar2	8		N	Must exist on ic_frgt_cls if supplied
price_class	varchar2	8		N	Must exist on ic_prce_cls if supplied
storage_class	varchar2	8		N	Must exist on ic_stor_cls if supplied
purch_class	varchar2	8		N	Must exist on ic_prch_cls if supplied
tax_class	varchar2	8		N	Must exist on ic_taxn_cls if supplied

Field/Column	Type	Length	Default	Req'd	Validation
customs_class	varchar2	8		N	Must exist on ic_ctms_cls if supplied
alloc_class	varchar2	8		N	Must exist on ic_allc_cls if supplied
planning_class	varchar2	8		N	Must exist on ps_plng_cls if supplied
itemcost_class	varchar2	8		N	Must exist on ic_cost_cls if supplied
cost_mthd_code	varchar2	4		N	Must exist on cm_mthd_mst if supplied
upc_code	varchar2	16		N	Not currently used
grade_ctl	number	5	0	N	0 = Not grade controlled 1 = Grade controlled Must be 0 if lot_ctl = 0
status_ctl	number	5	0	N	0 = Not status controlled 1 = Status controlled Must be 0 if lot_ctl = 0
qc_grade	varchar2	4		N	Must exist on qc_grad_mst if non-blank. Must be non-blank if grade_ctl = 1

Field/Column	Type	Length	Default	Req'd	Validation
lot_status	varchar2	4		N	Must exist on ic_lots_sts if non-blank. Must be non-blank if status_ctl = 1
bulk_id	number	10	0	N	Not currently used
pkg_id	number	10	0	N	Not currently used
qcitem_no	varchar2	32		N	Must exist on ic_item_mst if non-blank
qchold_res_code	varchar2	4		N	Must exist on qc_hrec_mst if non-blank
expaction_code	varchar2	4		N	Must exist on qc_actn_mst if non-blank
fill_qty	number		0	N	Not currently used
fill_um	varchar2	4		N	Not currently used
expaction_interval	number		0	N	Must not be negative value
phantom_type	number	5	0	N	
whse_item_no	varchar2	32		N	Must exist on ic_item_mst if non-blank
experimental_ind	number	5	0	N	0 = Non experimental item 1 = Experimental item
exported_date	date		01-Jan-1970 + 1 day	N	

Field/Column	Type	Length	Default	Req'd	Validation
seq_dpnd_class	varchar2	8		N	Must exist on cr_sqdt_cls if non-blank
commodity_code	varchar2	9		N	Must exist on ic_comd_cds if non-blank. Must be non-blank if GMI:Intrastat= "1"
ic_matr_days	number		0	N	Must not be negative value
ic_hold_days	number		0	N	Must not be negative value
attribute1	varchar2	240		N	Descriptive flexfield segment
attribute2	varchar2	240		N	Descriptive flexfield segment
attribute3	varchar2	240		N	Descriptive flexfield segment
attribute4	varchar2	240		N	Descriptive flexfield segment
attribute5	varchar2	240		N	Descriptive flexfield segment
attribute6	varchar2	240		N	Descriptive flexfield segment
attribute7	varchar2	240		N	Descriptive flexfield segment
attribute8	varchar2	240		N	Descriptive flexfield segment

Field/Column	Type	Length	Default	Req'd	Validation
attribute9	varchar2	240		N	Descriptive flexfield segment
attribute10	varchar2	240		N	Descriptive flexfield segment
attribute11	varchar2	240		N	Descriptive flexfield segment
attribute12	varchar2	240		N	Descriptive flexfield segment
attribute13	varchar2	240		N	Descriptive flexfield segment
attribute14	varchar2	240		N	Descriptive flexfield segment
attribute15	varchar2	240		N	Descriptive flexfield segment
attribute16	varchar2	240		N	Descriptive flexfield segment
attribute17	varchar2	240		N	Descriptive flexfield segment
attribute18	varchar2	240		N	Descriptive flexfield segment
attribute19	varchar2	240		N	Descriptive flexfield segment
attribute20	varchar2	240		N	Descriptive flexfield segment

Field/Column	Type	Length	Default	Req'd	Validation
attribute21	varchar2	240		N	Descriptive flexfield segment
attribute22	varchar2	240		N	Descriptive flexfield segment
attribute23	varchar2	240		N	Descriptive flexfield segment
attribute24	varchar2	240		N	Descriptive flexfield segment
attribute25	varchar2	240		N	Descriptive flexfield segment
attribute26	varchar2	240		N	Descriptive flexfield segment
attribute27	varchar2	240		N	Descriptive flexfield segment
attribute28	varchar2	240		N	Descriptive flexfield segment
attribute29	varchar2	240		N	Descriptive flexfield segment
attribute30	varchar2	240		N	Descriptive flexfield segment
attribute_category	varchar2	30		N	Descriptive flexfield structure defining column
user_name	varchar2	100	OPM	N	Ignored, but retained for backward compatibility.

Item Create API - Table and View Usage

The following OPM tables are referenced by the Item Master. The appropriate entries in these tables must exist and be non delete marked on the database for validation usage through the Item Create stored procedure.

Table Name	Select	Insert	Update	Delete	Base Table
ic_item_mst	X	X			
sy_uoms_mst	X				
sy_type_mst	X				
ic_invn_mst	X				
ic_rank_mst	X				
ic_gled_cls	X				
ic_invn_cls	X				
ic_sale_cls	X				
ic_ship_cls	X				
ic_frgt_cls	X				
ic_prce_cls	X				
ic_stor_cls	X				
ic_prch_cls	X				
ic_taxn_cls	X				
ic_ctms_cls	X				
ic_allc_cls	X				
ps_plng_cls	X				
qc_grad_mst	X				
qc_hres_mst	X				
qc_actn_mst	X				
ic_item_cpg		X			
ic_lots_mst		X			
ic_lots_sts	X				

The Item Master table IC_ITEM_MST holds all attributes of the item.

Lot controlled and quality control attributes are prompted for in the maintenance form once the appropriate indicators are set. These details are also held against the item.

Whether an item is defined as lot controlled or not the system always generates a DEFAULT lot against the item on IC_LOTS_MST.

All lots hold additional attributes for Consumer Packaged Goods (CPG)-specific processing. These are held against IC_LOTS_CPG.

When an item is identified as dual UOM and the alternate UOM is of a different type (for example, an item is defined in type WEIGHT with dual UOM in VOLUME) the save of the item master prompts for the conversion factor between WEIGHT and VOLUME for the item. This mechanism and structure supports transactional processing of the item in any supported UOM. This is handled outside of the Item Create procedure through an additional function.

Item Create API - Package/Procedure Names

The Item create API PL/SQL stored procedure code is held in the following:

- GMIPAPI

The stored procedure which is called to create a new inventory item is:

- Create_Item

Please refer to the Item Create API Wrapper for an example of how the above procedure is executed.

Item Create API - Special Logic

Initialization

Before calling this API, the GMIGUTL.setup procedure must be called to initialize various system constants.

Validation

No validation is applied to descriptive flexfield segments or to the user name.

Update Logic

When all the validation checks have been performed and no errors are found, a new item is created within the database. The following steps are followed:

1. Retrieve surrogate key item_id from sequence GEM5_ITEM_ID_S
2. Build row in ic_item_mst as follows:
   - All matching column names in PL/SQL record p_item_rec are transferred directly into the corresponding ic_item_mst columns.
   - The following columns on ic_item_mst require further explanation:

Column Name	Value
item_id	Surrogate key value as obtained above
qcitem_id	Item_id of p_item_rec.qcitem_no if not blank
whse_item_id	Item_id of p_item_rec.whse_item_no if not blank
created_by	Standard who column updated with FND_USER.USER_ID based on user_name supplied to the setup procedure
creation_date	System date
last_updated_by	Standard who column updated with FND_USER.USER_ID based on user_name supplied to the setup procedure
last_update_date	System date
last_update_login	Login id
trans_cnt	1 (first record update)
delete_mark	0 (not deleted)
text_code	Null (no text attached)

3. Build row in ic_item_cpg as follows if the SY$CPG_INSTALL flag =1

   - All matching column names in PL/SQL record p_item_rec are transferred directly into the corresponding ic_item_cpg columns.
   - The following columns on ic_item_cpg require further explanation:

Column Name	Value
item_id	Surrogate key value as obtained above
created_by	Standard who column updated with FND_USER.USER_ID based on user_name supplied to the setup procedure
creation_date	System date
last_updated_by	Standard who column updated with FND_USER.USER_ID based on user_name supplied to the setup procedure
last_update_date	System date
last_update_login	Login id

   - The default lot is now created in the lot master table (ic_lots_mst). This is achieved by calling the Create Lot API function (GMI_LOTS_PUB.Create_Lot). The standard API calling parameters and PL/SQL record are defined.
   - The p_init_msg parameter is set to false so that any messages generated by LotCreate is appended to any messages already generated.
   - As the committing of a new item is controlled by the Item Create API, the p_commit parameter is set to false for the purpose of calling the LotCreate API.

4. Build p_lot_rec PL/SQL record as follows:

Field/Column	Type	Length	Value
item_no	varchar2	32	p_item_rec.item_no
lot_no	varchar2	32	GMI:Default Lot
sublot_no	varchar2	32	
lot_desc	varchar2	40	
qc_grade	varchar2	4	

Field/Column	Type	Length	Value
expaction_code	varchar2	4	
expaction_date	date		01-Jan-1970 00:00:00
lot_created	date		01-Jan-1970 00:00:00
expire_date	date		31-Dec-2010 00:00:00
retest_date	date		system date/time
strength	number		100
inactive_ind	number	5	0
origination_type	number	5	0
shipvendor_no	varchar2	32	
vendor_lot_no	varchar2	32	
ic_matr_date	date		p_item_rec.ic_matr_days
ic_hold_date	date		p_item_rec.ic_hold_days
attribute1	varchar2	240	Descriptive flexfield segment
attribute2	varchar2	240	Descriptive flexfield segment
attribute3	varchar2	240	Descriptive flexfield segment
attribute4	varchar2	240	Descriptive flexfield segment
attribute5	varchar2	240	Descriptive flexfield segment
attribute6	varchar2	240	Descriptive flexfield segment
attribute7	varchar2	240	Descriptive flexfield segment
attribute8	varchar2	240	Descriptive flexfield segment
attribute9	varchar2	240	Descriptive flexfield segment

Field/Column	Type	Length	Value
attribute10	varchar2	240	Descriptive flexfield segment
attribute11	varchar2	240	Descriptive flexfield segment
attribute12	varchar2	240	Descriptive flexfield segment
attribute13	varchar2	240	Descriptive flexfield segment
attribute14	varchar2	240	Descriptive flexfield segment
attribute15	varchar2	240	Descriptive flexfield segment
attribute16	varchar2	240	Descriptive flexfield segment
attribute17	varchar2	240	Descriptive flexfield segment
attribute18	varchar2	240	Descriptive flexfield segment
attribute19	varchar2	240	Descriptive flexfield segment
attribute20	varchar2	240	Descriptive flexfield segment
attribute21	varchar2	240	Descriptive flexfield segment
attribute22	varchar2	240	Descriptive flexfield segment
attribute23	varchar2	240	Descriptive flexfield segment
attribute24	varchar2	240	Descriptive flexfield segment
attribute25	varchar2	240	Descriptive flexfield segment
attribute26	varchar2	240	Descriptive flexfield segment

Field/Column	Type	Length	Value
attribute27	varchar2	240	Descriptive flexfield segment
attribute28	varchar2	240	Descriptive flexfield segment
attribute29	varchar2	240	Descriptive flexfield segment
attribute30	varchar2	240	Descriptive flexfield segment
attribute_category	varchar2	30	Descriptive flexfield structure defining column
user_name	varchar2	100	Ignored, but retained for backward compatibility

Details of validation and update are contained in the topic on Lot Create API. If the lot create function is unsuccessful, the whole transaction is rolled back.

Item Create API - Error Messages

Listed below are all expected errors. These are output to the stored procedure message file and can be monitored through the return x_msg_count. This, in conjunction with the x_return_status can be used to monitor the success / fail of the procedure call.

Message Code	Narrative
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
IC_API_INVALID_SUBLOT_CTL	Invalid sublot 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

Message Code	Narrative
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
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

Message Code	Narrative
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

3

Item Create API Wrapper

This topic provides the business function, input structure, and ASCII flat file layout for the Item Create API Wrapper. The topic also presents package and procedure names, special logic, error messages, and a code example for Item Create API Wrapper.

The following topics are covered:

- Item Create API Wrapper - Business Function
- Item Create API Wrapper - Input Structure
- Item Create API Wrapper - ASCII Flat File Layout
- Item Create API Wrapper - Package and Procedure Names
- Item Create API Wrapper - Special Logic
- Item Create API Wrapper - Error Messages
- 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 which is used to create an inventory item in OPM. It may be required to be used in both synchronous (on-line) and asynchronous (batch) modes. When used in synchronous mode, the calling program (for example, an Oracle form) calls the API directly.

This topic is discusses using the API in asynchronous mode via a wrapper function. The source of data for the wrapper comes from:

- An ASCII flat file

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

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, but not necessarily, with a comma). The following table shows the order in which the data fields should appear and the maximum length of each data field.

Field Name	Type	Length	Req'd
item number	alphanumeric	32	Y
item description 1	alphanumeric	70	Y
item description 2	alphanumeric	70	N
alternative name for item	alphanumeric	32	N
second alternative name for item	alphanumeric	32	N
unit of measure	alphanumeric	4	Y
dual unit of measure indicator	number	5	Y
secondary unit of measure	alphanumeric	4	N
factor defining the allowable deviation below the std conversion for type 2/3	number	16	N
factor defining the allowable deviation above the std conversion for type 2/3	number	16	N
level code	number	5	N
lot controlled item indicator	number	1	Y
item is lot indivisible indicator	number	1	N
sub-lot controlled item indicator	number	1	N
location controlled item indicator	number	1	N
non-inventory item indicator	number	1	N
match type	number	5	N
inactive item indicator	number	1	N
inventory type	alphanumeric	4	N

Field Name	Type	Length	Req'd
shelf life	number	16	N
retest interval	number	16	N
item ABC code	alphanumeric	4	N
gl class	alphanumeric	8	N
inventory class	alphanumeric	8	N
sales class	alphanumeric	8	N
ship class	alphanumeric	8	N
freight class	alphanumeric	8	N
price class	alphanumeric	8	N
storage class	alphanumeric	8	N
purchase class	alphanumeric	8	N
tax class	alphanumeric	8	N
customs class	alphanumeric	8	N
allocation class	alphanumeric	8	N
planning class	alphanumeric	8	N
cost class	alphanumeric	8	N
cost method code (not currently used)	alphanumeric	4	N
upc code (not currently used)	alphanumeric	16	N
QC grade controlled item indicator	number	5	N
lot status controlled item indicator	number	5	N
default QC grade	alphanumeric	4	N
default lot status	alphanumeric	4	N
bulk id (not currently used)	number	10	N
pkg id (not currently used)	number	10	N
QC reference item	alphanumeric	32	N
QC hold reason code	alphanumeric	4	N
action code when a lot expires	alphanumeric	4	N
fill qty (not currently used)	number	16	N

Field Name	Type	Length	Req'd
fill um (not currently used)	alphanumeric	4	N
interval in days between when a lot expires and when the expiry action should be taken	number	16	N
phantom type (not currently used)	number	5	N
warehouse item number	alphanumeric	32	N
experimental item indicator	number	5	N
date item was exported to Oracle Financials	date (DDMMYYYY)	8	N
sequence dependent class	alphanumeric	8	N
commodity code	alphanumeric	9	N
lot maturity days	number	5	N
lot hold days	number	5	N
user name	alphanumeric	100	N
attribute1	alphanumeric	240	N
attribute2	alphanumeric	240	N
attribute3	alphanumeric	240	N
attribute4	alphanumeric	240	N
attribute5	alphanumeric	240	N
attribute6	alphanumeric	240	N
attribute7	alphanumeric	240	N
attribute8	alphanumeric	240	N
attribute9	alphanumeric	240	N
attribute10	alphanumeric	240	N
attribute11	alphanumeric	240	N
attribute12	alphanumeric	240	N
attribute13	alphanumeric	240	N
attribute14	alphanumeric	240	N
attribute15	alphanumeric	240	N

Field Name	Type	Length	Req'd
attribute16	alphanumeric	240	N
attribute17	alphanumeric	240	N
attribute18	alphanumeric	240	N
attribute19	alphanumeric	240	N
attribute20	alphanumeric	240	N
attribute21	alphanumeric	240	N
attribute22	alphanumeric	240	N
attribute23	alphanumeric	240	N
attribute24	alphanumeric	240	N
attribute25	alphanumeric	240	N
attribute26	alphanumeric	240	N
attribute27	alphanumeric	240	N
attribute28	alphanumeric	240	N
attribute29	alphanumeric	240	N
attribute30	alphanumeric	240	N
attribute category	alphanumeric	30	N

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

Item Create API Wrapper - Package and Procedure Names

The Item Create API wrapper PL/SQL stored procedure code is held in the following:

- GMI_ITEM_WRP

The procedure or function to be called to execute this API wrapper is:

- Create_Item

Item Create API Wrapper - Special Logic

Validation

Incorrectly formatted flat files are rejected.

The success or failure of the wrapper may be reported back to the calling function by means of the return value. This is hierarchically as follows:

- On initial entry to the wrapper the return status are set to success ('S')
- If for any record processed an expected error occurs and the return status is currently set to success then it are updated to expected error ('E').
- If for any record processed and unexpected error occurs, then the return status is set to unexpected error ('U').

Update Logic

Updates are only be concerned with the processing of messages (errors and others) generated by the item create API.

Messages (success and error) are written to a flat file designated by the p_output_ file parameter. A log file is written to the /tmp directory. This details start and completion times, data retrieved from the ASCII flat file and messages generated.

Item Create API Wrapper - Error Messages

The errors listed below may be generated by the API wrapper. These messages are generally related to the handling of the ASCII flat input and output files. These messages are sent to the standard output device since it is inappropriate to attempt to send them to the files that themselves may be causing the erroneous condition. They are hard-coded in English.

Error Condition	Narrative
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

Item Create API Wrapper - Code Example

The PL/SQL code for this API wrapper is as follows:

```
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
--

dbms_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,'frt_class      = '||item_rec.frt_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;
```

```
WHEN UTL_FILE.WRITE_ERROR THEN
   dbms_output.put_line('Invalid Write Error   '|| l_global_file);
   UTL_FILE.FCLOSE_ALL;
   RETURN l_return_status;

WHEN UTL_FILE.READ_ERROR THEN
   dbms_output.put_line('Invalid Read  Error   '|| l_global_file);
   UTL_FILE.FCLOSE_ALL;
   RETURN l_return_status;

WHEN UTL_FILE.INTERNAL_ERROR THEN
   dbms_output.put_line('Internal Error');
   UTL_FILE.FCLOSE_ALL;
   RETURN l_return_status;

WHEN OTHERS THEN
   dbms_output.put_line('Other Error');
   UTL_FILE.FCLOSE_ALL;
   RETURN l_return_status;

END Create_Item;

--+=========================================================================+
--| FUNCTION NAME                                                           |
--|    Get_Field                                                            |
--|                                                                         |
--| TYPE                                                                    |
--|    Public                                                               |
--|                                                                         |
--| USAGE                                                                   |
--|    Get value of field n from a delimited line of ASCII data             |
--|                                                                         |
--| DESCRIPTION                                                             |
--|    This utility function will return the value of a field from          |
--|    a delimited line of ASCII text                                       |
--|                                                                         |
--| PARAMETERS                                                              |
--|    p_line             IN VARCHAR2         - line of data                |
--|    p_delimiter        IN VARCHAR2         - Delimiter character         |
--|    p_field_no         IN NUMBER           - Field occurance to be       |
--|                                             returned                    |
--|                                                                         |
--| RETURNS                                                                 |
--|    VARCHAR2                               - Value of field              |
--|                                                                         |
```

```
--| 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;
```

4

Item Lot/Sublot Conversion API

This topic provides the business function and technical overview for the Item Lot / Sublot Conversion API. Parameters for Item Lot / Sublot conversion are included with information on table and view use, and package and procedure names. The topic also provides information on special logic, error messages, and success messages.

The following topics are covered:

- Item Lot / Sublot Conversion API - Business Function
- Item Lot / Sublot Conversion API - Technical Overview
- Item Lot / Sublot Conversion API - Parameters
- Item Lot / Sublot Conversion API - Table and View Usage
- Item Lot / Sublot Conversion API - Package and Procedure Names
- Item Lot / Sublot Conversion API - Special Logic
- Item Lot / Sublot Conversion API - Error Messages
- Item Lot / Sublot Conversion API - Success Messages

Item Lot/Sublot Conversion API - Business Function

This stored procedure is concerned with the following function within the Oracle Process Manufacturing (OPM) Inventory Management Module:

- Create an item lot / sublot unit of measure conversion

The document describes how the stored procedure should be called, the parameters that are required (and optional) and the values that are returned to the calling program. This includes all error conditions that may arise.

The procedure 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 - Technical Overview

The Item Lot Conversion stored procedure is intended to be used by a user 'wrapper' calling function with item attributes passed to the procedure via a RECORD format which is detailed in this document. 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.

Item Lot/Sublot Conversion API - Parameters

There are two variants of this procedure which 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  GMIGAPI.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 4, and last 3 parameters are standard across all of the API calls and are identical to the Create Item API, where they are fully documented. If the creation is successful, 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. If this is the case then p_ic_item_mst_row and p_ic_lots_mst_row should be 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.

This p_conv_rec parameter passes the item-specific data required to create an inventory item. It is described in the following. Please refer to the section on the Item Lot/Sublot Conversion API Wrapper for an example of how to populate and call the stored procedure.

Field/Column	Type	Length	Default	Req'd	Validation
item_no	varchar2	32		Y	Must exist on ic_item_mst Must not be deleted Must be active
lot_no	varchar2	32		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	varchar2	32		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	varchar2	4		Y	Must exist on sy_uoms_mst Must be of same UOM type as ic_item_mst.item_um
to_uom	varchar2	4		Y	Must exist on sy_uoms_mst Must be of different UOM type to ic_item_mst.item_um

Field/Column	Type	Length	Default	Req'd	Validation
type_factor	number			Y	Conversion factor. 1 from-uom equals this number of to_uom
user_name	varchar2	100	'OPM'	N	Ignored but retained for backward compatibility

Item Lot/Sublot Conversion API - Table and View Usage

The following OPM tables are referenced by the Item Lot Conversion API. The appropriate entries in these tables must exist and be non-delete marked on the database for validation usage through the Item Create stored procedure.

Table Name	Select	Insert	Update	Delete	Base Table
ic_item_mst	X				
sy_uoms_mst	X				
ic_item_cnv	X	X	X		
sy_uoms_typ	X				
ic_lots_mst	X				

The Item Lot UOM conversion table IC_ITEM_CNV holds unit of measure conversion values for the item / lot.

When an item is identified as dual UOM and the alternate UOM is of a different type (for example, an item is defined in type WEIGHT with dual UOM in VOLUME) the save of the item master prompts for the conversion factor between WEIGHT and VOLUME for the item. This mechanism and structure supports transactional processing of the item in any supported UOM.

Item Lot/Sublot Conversion API - Package and Procedure Names

The Item Lot / Sublot conversion API PL / SQL stored procedure code is held in the following:

- GMIPAPI

The stored procedure which is called to create a new Item Lot / Sublot conversion is:

- Create_Lot

Please refer to the Item Lot / Sublot conversion API wrapper specification for an example of how the above procedure is executed.

Item Lot/Sublot Conversion API - Special Logic

Validation

The validation is described in p_item_cnv_rec in the Item Lot / Sublot Conversion API - Parameters topic.

Update Logic

When all the validation checks have been performed and no errors are found, a new item / lot conversion is created within the database. The following steps are followed:

1. Get unit of measure type of from_um from sy_uoms_mst. (A)
2. Get unit of measure type of to_um for sy_uoms_mst (B)
3. Get unit of measure type of primary unit of measure for item (C)
4. If A=C then p_item_cnv_rec.conv_factor = 1 / p_item_cnv_rec.conv_factor.
5. Get standard unit of measure for (A) from sy_uoms_typ.
6. If standard unit of measure differs from the from unit of measure then use the standard unit of measure conversion routine to convert from p_item_cnv_ rec.from_um to sy_uoms_typ.std_um.
7. Get standard unit of measure for (B) from sy_uoms_typ.
8. If standard unit of measure differs from the to unit of measure then use the standard unit of measure conversion routine to convert from p_item_cnv_ rec.to_um to sy_uoms_typ.std_um.
9. Build a row in ic_item_cnv as given in the following table:

Column Name	Value
item_id	ic_item_mst.item_id
lot_id	ic_lots_mst.lot_id or zero if lot_no not supplied
um_type	sy_uoms_typ.um_type of p_item_cnv_rec.to_um
type_factor	Calculated above
creation_date	System date
last_update_date	System date
created_by	FND_USER.user_id based on supplied user_name
last_updated_by	FND_USER.user_id based on supplied user_name
trans_cnt	1
delete_mark	0
text_code	0
type_factorrev	Reciprocal of type_factor
last_update_login	login_id

Item Lot/Sublot Conversion API - Error Messages

Listed below are all expected errors. These are output to the stored procedure message file and can be monitored through the return x_msg_count. This, in conjunction with the x_return_status can be used to monitor the success or failure of the procedure call.

Message Code	Narrative
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
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

Item Lot/Sublot Conversion API - Success Messages

If the Item Lot / Sublot conversion create is successful then the API returns a success message as follows:

Message Code	Narrative
IC_API_ILC_CREATED	Conversion created for item &ITEM_NO lot &LOT_NO - SUBLOT_NO UOM type &UM_TYPE

Translation of error messages is determined by the value passed to the API in p_ item_cnv_rec.op_code. If this is not supplied then this defaults to the value 'API'. The default language for the op_code determines the language in which the messages are translated. If the op_code does not exist on sy_oper_mst or the message is not found in the required language, then the message are retrieved in English (lang_code = 'ENG').

The 'API' operator are inserted into sy_oper_mst by the installation script with a default language code of 'ENG'.

5

Item Lot/Sublot Conversion API Wrapper

This topic provides the business function, technical requirements, input structure, special logic, ASCII flat file layout, and package and procedure names for the Item Lot / Sublot Conversion API Wrapper. Error messages and a code example are also supplied.

The following topics are covered:

- Item Lot / Sublot Conversion API Wrapper - Business Function
- Item Lot / Sublot Conversion API Wrapper - Technical Requirements
- Item Lot / Sublot Conversion API Wrapper - Input Structure
- Item Lot / Sublot Conversion API Wrapper - Special Logic
- Item Lot / Sublot Conversion API Wrapper - ASCII Flat File Layout
- Item Lot / Sublot Conversion API Wrapper - Package and Procedure Names
- Item Lot / Sublot Conversion API Wrapper - Error Messages
- Item Lot / Sublot Conversion API Wrapper - Code Example

Item Lot/Sublot Conversion API Wrapper - Business Function

This stored procedure is designed to operate in conjunction with the item/lot unit of measure conversion API. This API is used to create an item/lot unit of measure conversion within OPM. It may be required to be used in both synchronous (that is, on-line) and asynchronous (that is, batch) modes. When used in synchronous mode, the calling program (for example, an Oracle form) calls the API directly.

This specification is concerned with using the API in asynchronous mode using a wrapper function. The source of data for the wrapper comes from:

- An ASCII flat file

This topic describes how the wrapper function should be called and the parameter requirements.

Item Lot/Sublot Conversion API Wrapper - Technical Requirements

The Item Lot Conversion API stored procedure is intended to be used by a user 'wrapper' calling function with item/lot unit of measure conversion attributes passed to the procedure via a RECORD format which is detailed in this topic. The wrapper passes the appropriate parameters into the stored API procedure and is responsible for handling the return code from the procedure and any associated messages.

Item Lot/Sublot Conversion API Wrapper - Input Structure

The API wrapper consists of a PL/SQL procedure and PL/SQL function both named 'Create_Conv'.

Item Lot/Sublot Conversion API Wrapper - Special Logic

Validation

Incorrectly formatted flat files are rejected.

The success or failure of the wrapper may be reported back to the calling function by means of the return value. This is hierarchically as follows:

- On initial entry to the wrapper the return status is set to success ('S').
- If for any record processed an expected error occurs and the return status is currently set to success then it is updated to expected error ('E').

- If for any record processed and unexpected error occurs, then the return status is set to unexpected error ('U').

Update Logic

Updates are only concerned with the processing of messages (errors and others) generated by the item/lot conversion API.

Messages (success and error) are written to a flat file as designated by the p_output_file parameter. Additionally a log file is written to the /tmp directory. This details the start and completion times, data retrieved from the ASCII flat file, and messages generated.

Item Lot/Sublot Conversion API Wrapper - ASCII Flat File Layout

The ASCII flat file may be 'character delimited' (typically, but not necessarily, with a comma) or it can be in fixed position format. The table below shows the order in which the data fields must appear (for delimited files) and the length of each data field (for fixed position format files).

Field Name	Type	Length	Required
item number	alphanumeric	32	Y
lot number	alphanumeric	32	N
sublot number	alphanumeric	32	N
from unit of measure	alphanumeric	4	Y
to unit of measure	alphanumeric	4	Y
conversion factor	number	16	Y
user name	alphanumeric	100	N

Use consecutive delimiters to omit an optional field.

Item Lot/Sublot Conversion API Wrapper - Package and Procedure Names

The Item Lot/Sublot conversion API wrapper PL/SQL stored procedures code are held in the package

- GMI_ITEM_LOT_CONV_WRP

The procedure or function to be called to execute this API wrapper is:

- Create_Conv

Item Lot/Sublot Conversion API Wrapper - Error Messages

Error messages generated by the Item Lot/Sublot Conversion API are written to the file 'p_output_file'.

Errors listed below may be generated by the API wrapper. These messages are generally related to the handling of the ASCII flat input and output files. These messages are sent to the standard output device as it is inappropriate to attempt to send them to the files which themselves may be causing the erroneous condition. They are hard-coded in English.

Error Condition	Narrative
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

Item Lot/Sublot Conversion API Wrapper - Code Example

The PL/SQL code for this API wrapper is as follows:

```
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
CREATE OR REPLACE PACKAGE BODY GMI_ITEM_LOT_CONV_WRP AS
-- $Header: GMIPILWB.pls 115.5 2000/08/10 15:00:08 hverddin gmigapib.pls $
--+=========================================================================+
--| PROCEDURE NAME                                                          |
--|    Create_conv                                                          |
--|                                                                         |
--| TYPE                                                                    |
--|    Public                                                               |
--|                                                                         |
--| USAGE                                                                   |
--|    Create an Item/Lot/Sublot UoM conversion                             |
--|                                                                         |
--| DESCRIPTION                                                             |
--|    This is a PL/SQL wrapper procedure to call the Create_Conv           |
--|    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                                                                 |
--|                                                                         |
--+=========================================================================+
-- Api end of comments
PROCEDURE Create_Conv
( 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_conv( p_dir
     , p_input_file
     , p_output_file
     , p_delimiter
     );

End Create_Conv;

--+=========================================================================+
--| FUNCTION NAME                                                           |
--|    Create_conv                                                          |
--|                                                                         |
--| TYPE                                                                    |
--|    Public                                                               |
--|                                                                         |
--| USAGE                                                                   |
--|    Create an Item/Lot/Sublot UoM conversion                             |
--|                                                                         |
--| DESCRIPTION                                                             |
--|    This is a PL/SQL wrapper function to call the OPM                    |
--|    Inventory Item Lot/Sublot UOM Conversion Create 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 /Out 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_Conv
( 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_dummy_cnt          NUMBER  :=0;
l_loop_cnt           NUMBER  :=0;
l_record_count       NUMBER  :=0;
l_data               VARCHAR2(2000);
item_cnv_rec         GMIGAPI.conv_rec_typ;
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(200);
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);
l_ic_item_cnv_row    ic_item_cnv%ROWTYPE;

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 Item Lot/Sublot UOM Conversion create API
--

UTL_FILE.PUT_LINE(l_log_handle, 'Process Started at '
|| to_char(SYSDATE,'DD-MON-YY HH: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_cnv_rec.item_no      :=Get_Field(l_line,l_delimiter,1);
  item_cnv_rec.lot_no       :=Get_Field(l_line,l_delimiter,2);
  item_cnv_rec.sublot_no    :=Get_Field(l_line,l_delimiter,3);
  item_cnv_rec.from_uom     :=Get_Field(l_line,l_delimiter,4);
  item_cnv_rec.to_uom       :=Get_Field(l_line,l_delimiter,5);
  IF (Get_Field(l_line,l_delimiter,6) IS NULL)
  THEN
    item_cnv_rec.type_factor:=0;
  ELSE
    item_cnv_rec.type_factor:=TO_NUMBER(Get_Field(l_line,l_delimiter,6));
  END IF;

  IF (Get_Field(l_line,l_delimiter,7) IS NULL)
  THEN
    item_cnv_rec.user_name       :='OPM';
  ELSE
    item_cnv_rec.user_name       :=Get_Field(l_line,l_delimiter,7);
  END IF;

UTL_FILE.PUT_LINE(l_log_handle,'item no    = '||item_cnv_rec.item_no );
UTL_FILE.PUT_LINE(l_log_handle,'lot no     = '||item_cnv_rec.lot_no );
UTL_FILE.PUT_LINE(l_log_handle,'sublot no  = '||item_cnv_rec.sublot_no );
UTL_FILE.PUT_LINE(l_log_handle,'from_uom   = '||item_cnv_rec.from_uom );
UTL_FILE.PUT_LINE(l_log_handle,'to_uom     = '||item_cnv_rec.to_uom );
UTL_FILE.PUT_LINE(l_log_handle,'type_factor= '||item_cnv_rec.type_factor );
UTL_FILE.PUT_LINE(l_log_handle,'op Code    = '||item_cnv_rec.user_name );

dbms_output.put_line('Calling creation routine');

GMIPAPI.Create_Item_Lot_Conv
( 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_conv_rec       => item_cnv_rec
, x_ic_item_cnv_row => l_ic_item_cnv_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 HH: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 FND_API.G_RET_STS_UNEXP_ERROR;

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

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

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

WHEN UTL_FILE.WRITE_ERROR THEN
   dbms_output.put_line('Invalid Write Error   '|| l_global_file);
   UTL_FILE.FCLOSE_ALL;
   RETURN FND_API.G_RET_STS_UNEXP_ERROR;

WHEN UTL_FILE.READ_ERROR THEN
   dbms_output.put_line('Invalid Read  Error   '|| l_global_file);
   UTL_FILE.FCLOSE_ALL;
   RETURN FND_API.G_RET_STS_UNEXP_ERROR;
```

```
WHEN UTL_FILE.INTERNAL_ERROR THEN
   dbms_output.put_line('Internal Error');
   UTL_FILE.FCLOSE_ALL;
   RETURN FND_API.G_RET_STS_UNEXP_ERROR;

WHEN OTHERS THEN
   dbms_output.put_line('Other Error');
   UTL_FILE.FCLOSE_ALL;
   RETURN FND_API.G_RET_STS_UNEXP_ERROR;

END Create_Conv;

--+=========================================================================+
--| FUNCTION NAME                                                           |
--|    Get_Field                                                            |
--|                                                                         |
--| TYPE                                                                    |
--|    Public                                                               |
--|                                                                         |
--| USAGE                                                                   |
--|    Get value of field n from a delimited line of ASCII data             |
--|                                                                         |
--| DESCRIPTION                                                             |
--|    This utility function will return the value of a field from          |
--|    a delimited line of ASCII text                                       |
--|                                                                         |
--| PARAMETERS                                                              |
--|    p_line             IN VARCHAR2         - line of data                |
--|    p_delimiter        IN VARCHAR2         - Delimiter character         |
--|    p_field_no         IN NUMBER           - Field occurance to be       |
--|                                             returned                    |
--|                                                                         |
--| RETURNS                                                                 |
--|    VARCHAR2                               - Value of field              |
--|                                                                         |
--| 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_LOT_CONV_WRP;
/
commit;
exit;
```

6

Inventory Quantities API

This topic provides the business function and technical overview for the Inventory Quantities API. Parameters for Inventory Quantities API are included with information on table and view use, and package and procedure names. The topic also provides information on special logic, error messages, and success messages.

The following topics are covered:

- Inventory Quantities API - Business Function
- Inventory Quantities API - Technical Overview
- Inventory Quantities API - Parameters
- Inventory Quantities API - Table and View Usage
- Inventory Quantities API - Package and Procedure Names
- Inventory Quantities API - Special Logic
- Inventory Quantities API - Error Messages
- Inventory Quantities API - Success Messages

Inventory Quantities API - Business Function

This stored procedure is concerned with the following functions within the Oracle Process Manufacturing (OPM) Inventory Management module:

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

This topic describes how the stored procedure should be called, the parameters that are required (and optional) and the values that are returned to the calling program. This includes all error conditions that may arise.

Inventory Quantities API - Technical Overview

The Inventory Quantities stored procedure is intended to be used by a user 'wrapper' calling function with item attributes passed to the procedure via a RECORD format which is detailed in this topic. 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.

Inventory Quantities API - 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 they 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  GMIGAPI.qty_rec_typ
, 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 4, and last 3 parameters are standard across all of the API calls and are identical to the Create Item API where they are fully documented. If the posting is successful, 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 returned s 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.

The p_qty_rec parameter is used to pass the item-specific data required to create an inventory item. It is described below. Please refer to the Inventory Quantities API Wrapper topic for an example of how to populate this parameter and call the stored procedure.

Field/Column	Type	Length	Default	Req'd	Validation
trans_type	number	2		Y	1 - create inventory 2 - adjust inventory 3 - move inventory 4 - change lot status 5 - change QC grade No other values allowed
item_no	varchar2	32		Y	Must exist on ic_item_mst Must not be deleted May be inactive if the IC$ALLOW_INACTIVE flag is set to 1, must not be noninventory
journal_no	varchar2	32		N	Must be blank if automatic document sequencing Must not exist in ic_jrnl_mst if manual document sequencing

Field/Column	Type	Length	Default	Req'd	Validation
from_whse_ code	varchar 2	4		Y	May be blank if trans_ type = 5 Must exist on ic_whse_ mst for all other transaction types
to_whse_code	varchar 2	4	NULL	N	Must exist on ic_whse_ mst if trans_type = 3
item_um	varchar 2	4		Y	Must exist on sy_uoms_ mst Must be of same UOM type as primary UOM of item
item_um2	varchar 2	4		Y	Must exist on sy_uoms_ mst Must be of same UOM type as secondary UOM of item
lot_no	varchar 2	32	blank	N	Must be blank if ic_item_ mst.lot_ctl = 0 Must not equal GMI:Default Lot
sublot_no	varchar 2	32	blank	N	Must be blank if ic_item_ mst.sublot_ctl = 0
from_location	varchar 2	16		N	See Validation topic
to_location	varchar 2	16		N	See Validation topic
trans_qty	number			N	See Validation topic
trans_qty2	number			N	See Validation topic
qc_grade	varchar 2	4		N	Must exist on qc_grad_ mst if supplied
lot_status	varchar 2	4		N	Must exist on ic_lots_mst if supplied
co_code	varchar 2	4		Y	Must exist in sy_orgn_mst

Field/Column	Type	Length	Default	Req'd	Validation
orgn_code	varchar 2	4			Must exist in sy_orgn_mst and sy_orgn_mst.co_code must be co_code above
trans_date	date			Y	Must be in an open inventory calendar
reason_code	varchar 2	4		Y	Must exist in sy_reas_cds
user_name	varchar 2	100	'OPM'	N	Ignored but retained for backward compatibility

Inventory Quantities API - Table and View Usage

The following OPM tables are referenced by the Inventory Quantities API. The appropriate entries in these tables must exist and be non-delete marked on the database for validation usage through the Item Create stored procedure.

Table Name	Select	Insert	Update	Delete	Base Table
ic_item_mst	X				
sy_uoms_mst	X				
ic_item_cnv	X	X	X		
sy_uoms_typ	X				
ic_lots_mst	X		X		
ic_jrnl_mst		X			
ic_adjs_jnl		X			
ic_tran_cmp		X			
ic_loct_inv	X	X	X		
ic_summ_inv		X	X		
ic_lots_sts	X				

The Item Lot UOM conversion table IC_ITEM_CNV holds unit of measure conversion values for the item / lot.

When an item is identified as dual UOM and the alternate UOM is of a different type (for example, the item is defined in type WEIGHT with dual UOM in VOLUME) the save of the item master prompts for the conversion factor between WEIGHT and VOLUME for the item. This mechanism and structure supports transactional processing of the item in any supported UOM.

Inventory Quantities API - Package and Procedure Names

The Inventory Quantities API PL/SQL stored procedure code is held in the following:

GMIPAPI

The stored procedure which is called to post an inventory transaction is:

- Inventory_Posting

Please refer to the Inventory Quantities API Wrapper topic for an example of how the previous procedure is executed.

Inventory Quantities API - Special Logic

Validation

In addition to the validation described in p_trans_rec table the following validation is performed. Certain validation is dependent on type of transaction (p_trans_rec.trans_type).

- For grade change transactions (trans_type=5) the item is validated to be grade controlled.
- For lot status change transactions (trans_type=4) the item is validated to be status controlled.
- If the item is lot-controlled or lot/sublot-controlled then for create inventory transactions (trans_type=1) the item/lot/sublot may not exist. For all other transaction types the item/lot/sublot must exist.
- For all transactions except change (QC) grade, the "from warehouse" is validated and the "from location" is validated according to the item/warehouse location rules.
- For move inventory transactions, the "to warehouse" is validated and the "to location" is similarly validated.
- For transactions affecting quantity for dual unit of measure items, the secondary quantity is validated to be within tolerance of the primary quantity.
- For adjustments, movements and change lot status, the inventory location is validated to ensure an on-hand balance exists at the location. For create inventory transactions, there should be no location balance.

- If negative inventory is not allowed (GMI:Allow Negative Inventory = 0) then the on-hand balance at the location is validated to ensure that sufficient stock exists such that the balance will not become negative after the transaction has been updated.
- If moving inventory to a location where inventory already exists with a different lot status, and this is not allowed (GMI:Move Different Status), then an error is generated.
- Inactive items may be processed if the IC$ALLOW_INACTIVE flag is set to 1.

Update Logic

When all the validation checks have been performed and no errors found, the transaction can be posted. The following steps are followed:

1. The surrogate key (journal_id) is obtained from the surrogate key generator and a row is inserted in the journal header table as follows:
2. Insert journal header ic_jrnl_mst as follows:

Column Name	Value
journal_id	Surrogate key value
journal_no	If automatic document numbering then document number from sy_docs_seq where orgn_code= p_trans_rec.orgn_code and doc_type='JRNL' If manual numbering then p_trans_rec.journal_no
journal_comment	Blank
posting_id	Zero
print_cnt	Zero
posted_ind	1
orgn_code	from p_trans_rec.orgn_code
creation_date	system date
last_update_date	system date
created_by	FND_USER.user_id based on supplied user_name
last_updated_by	FND_USER.user_id based on supplied user_name

Column Name	Value
program_ update_date	System date
last_update_ login	login_id
in_use	0
delete_mark	0
text_code	Null

3. The inventory adjustment detail table is updated:
   - For Create inventory and adjust inventory transactions, one row is inserted in the inventory adjustment detail table.
   - For Move inventory, lot status and (QC) grade change transactions, two rows are inserted. This value is stored as a loop counter.
4. First the surrogate key (doc_id) is obtained the sequence gem5_doc_id_s
5. The remainder of the update processing is performed either once or twice depending on the value of the loop counter previously described.
6. Get the surrogate key (line_id) from sequence gem5_line_id_s
7. Insert the following record into the ic_adjs_jnl table:

Column Name	Value
trans_type	CREI if p_trans_rec.trans_type = 1 ADJI if p_trans_rec.trans_type = 2 TRNI if p_trans_rec.trans_type = 3 STSI if p_trans_rec.trans_type = 4 GRDI if p_trans_rec.trans_type = 5
trans_flag	Zero
doc_id	Surrogate key value obtained above
doc_line	Loop counter
journal_id	Same as ic_jrnl_mst.journal_id (see above)
completed_ind	1

Column Name	Value
whse_code	If doc_line = 1 then p_trans_rec.from_whse_code If doc_line = 2 then p_trans_rec.to_whse_code
reason_code	from p_trans_rec.reason_code
doc_date	from p_trans_rec.trans_date
item_id	ic_item_mst.item_id where item_no=p_trans_rec.item_no
item_um	p_trans_rec.item_um
item_um2	p_trans_rec.item_um2
lot_id	ic_lots_mst.lot_id where item_no/lot_no/sublot_no from p_trans_rec
location	If doc_line = 1 then p_trans_rec.from_location If doc_line = 2 then p_trans_rec.to_location
qty	p_trans_rec.trans_qty
qty2	p_trans_rec.trans_qty2
qc_grade	If doc_line = 1 then ic_lots_mst.qc_grade If doc_line =2 then p_trans_rec.qc_grade
lot_status	for doc_line 1 then ic_loct_inv.lot_status for doc_line 2 then p_trans_rec.lot_status
line_type	If trans_type = 1 or trans_type =2 then 0 Else If doc_line = 1 then -1 If doc_line = 2 then 1
line_id	Surrogate key value obtained above
co_code	p_trans_rec.co_code
orgn_code	p_trans_rec.orgn_code
no_inv	Zero
no_trans	Zero
creation_date	System date
last_update_date	System date
created_by	FND_USER.user_id based on supplied user_name

Column Name	Value
last_updated_by	FND_USER.user_id based on supplied user_name
program_update_ date	System date
trans_cnt	1
last_update_login	login_id

8. For each row inserted into the inventory adjustment detail table, the PL/SQL record p_cmp_tran_rec defined below is populated and the Completed Inventory Transaction Processor is called.

Completed Inventory Transaction Processor

The completed inventory transaction processor performs the following updates all data required is supplied in the PL/SQL record type p_cmp_tran_rec documented previously.

All Transaction Types

A row is inserted in the completed inventory transaction table.

If the surrogate key (trans_id) supplied is zero then the next value is obtained from the surrogate key sequence gem5_trans_id_s.

ic_tran_cmp

Column Name	Value
item_id	p_cmp_tran_rec.item_id
line_id	p_cmp_tran_rec.line_id
trans_id	surrogate key obtained above
co_code	p_cmp_tran_rec.co_code
orgn_code	p_cmp_tran_rec.orgn_code
whse_code	p_cmp_tran_rec.whse_code
lot_id	p_cmp_tran_rec.lot_id
location	p_cmp_tran_rec.location
doc_id	p_cmp_tran_rec.doc_id

Column Name	Value
doc_type	p_cmp_tran_rec.trans_type
doc_line	p_cmp_tran_rec.doc_line
line_type	p_cmp_tran_rec.line_type
reason_code	p_cmp_tran_rec.reason_code
trans_date	p_cmp_tran_rec.doc_date
trans_qty	p_cmp_tran_rec.trans_qty
trans_qty2	p_cmp_tran_rec.trans_qty2
qc_grade	p_cmp_tran_rec.qc_grade
lot_status	p_cmp_tran_rec.lot_status
trans_stat	p_cmp_tran_rec.trans_stat
trans_um	p_cmp_tran_rec.trans_um
trans_um2	p_cmp_tran_rec.trans_um2
op_code	p_cmp_tran_rec.op_code
gl_posted_ind	p_cmp_tran_rec.gl_posted_ind
event_id	p_cmp_tran_rec.event_id
text_code	p_cmp_tran_rec.text_code
creation_date	system date
last_update_date	system date
created_by	FND_USER.user_id based on supplied user_name
last_updated_by	FND_USER.user_id based on supplied user_name
last_update_login	login_id

Inventory Movement Transactions

If the transaction is a create (p_cmp_tran_rec.doc_type = CREI), an adjust (p_cmp_tran_rec.doc_type = ADJI) or a move inventory (p_cmp_tran_rec.doc_type = TRNI) then the ic_loct_inv location inventory table is updated as follows:

where item_id = p_cmp_tran_rec.item_id

and whse_code = p_cmp_tran_rec.whse_code

and lot_id = p_cmp_tran_rec.tran_id

and location = p_cmp_tran_rec.location

Column Name	Value
loct_onhand	loct_onhand + p_cmp_tran_rec.trans_qty
loct_onhand2	loct_onhand2 + p_cmp_tran_rec.trans_qty2

The Inventory summary table is then updated. The lot status table is read (ic_lots_sts.lot_status = p_cmp_tran_rec.lot_status) in order to obtain the lot status indicators. The ic_summ_inv table is then updated as follows:

where item_id = p_cmp_tran_rec.item_id

and whse_code = p_cmp_tran_rec.whse_code

and qc_grade = p_cmp_tran_rec.qc_grade

Column Name	Value
onhand_qty	if ic_lots_sts.nettable_ind = 1 then onhand_qty = onhand_qty + p_cmp_tran_rec.trans_qty
onhand_qty2	if ic_lots_sts.nettable_ind = 1 then onhand_qty2 = onhand_qty2 + p_cmp_tran_rec.trans_qty2
onhand_prod_qty	if ic_lots_sts.prod_ind = 1 then onhand_prod_qty = onhand_prod_qty + p_cmp_tran_rec.trans_qty
onhand_prod_qty2	if ic_lots_sts.prod_ind = 1 then onhand_prod_qty2= onhand_prod_qty2 + p_cmp_tran_rec.trans_qty2
onhand_order_qty	if ic_lots_sts.order_proc_ind = 1 then onhand_order_qty = onhand_order_qty + p_cmp_tran_rec.trans_qty
onhand_order_qty2	if ic_lots_sts.order_proc_ind = 1 then onhand_order_qty2 = onhand_order_qty2 + p_cmp_tran_rec.trans_qty2
onhand_ship_qty	if ic_lots_sts.shipping_ind = 1 then onhand_ship_qty = onhand_ship_qty + p_cmp_tran_rec.trans_qty
onhand_ship_qty2	if ic_lots_sts.shipping_ind = 1 then onhand_ship_qty2 = onhand_ship_qty2 + p_cmp_tran_rec.trans_qty2

loct_onhand2	loct_onhand2 + p_cmp_tran_rec.trans_qty2

If the row is not found in the inventory summary table then it is created with zero value quantities prior to be updated as above.

Change Lot Status Transactions

For a change lot status transaction the on-hand quantities must be obtained from the inventory location table (ic_loct_inv). These are used to populate the PL/SQL record p_cmp_tran_rec as follows:

If p_cmp_tran_rec.line_type = -1 then p_cmp_tran_rec.trans_qty = 0 - ic_loct_inv.loct_onhand, p_cmp_tran_rec.trans_qty2 = 0 - ic_loct_inv.loct_onhand2

Else p_cmp_tran_rec.trans_qty = ic_loct_inv.loct_onhand, p_cmp_tran_rec.onhand2 = ic_loct_inv.loct_onhand2

For the 'after' transaction (p_cmp_tran_rec.line_type = 1) then the lot status is updated in ic_loct_inv,

where item_id = p_cmp_tran_rec.item_id

and whse_code = p_cmp_tran_rec.whse_code

and lot_id = p_cmp_tran_rec.tran_id

and location = p_cmp_tran_rec.location

Column Name	Value
lot_status	p_cmp_tran_rec.lot_status

The transaction quantity values are then used to update the summary inventory table (ic_summ_inv) as detailed for inventory movements above.

Change QC Grade transactions

For a change QC grade transaction, the QC grade is updated on the lots master table (ic_lots_mst) as follows:

Column Name	Value
qc_grade	If p_cmp_tran_rec.line_type = 1 then p_cmp_tran_rec.qc_grade

1. The warehouses at which the lot exists and their on-hand quantities must be obtained from the inventory location table (ic_loct_inv). This are used to populate the PL/SQL record p_cmp_tran_rec as follows:
   - If p_cmp_tran_rec.line_type = -1 then p_cmp_tran_rec.whse_code = ic_loct_inv.whse_code, p_cmp_tran_rec.trans_qty = 0 - SUM(ic_loct_inv.loct_onhand), p_cmp_tran_rec.trans_qty2 = 0 - SUM(ic_loct_inv.loct_onhand2)
   - Else p_cmp_tran_rec.whse_code = ic_loct_inv.whse_code, p_cmp_tran_rec.trans_qty = SUM(ic_loct_inv.loct_onhand), p_cmp_tran_rec.onhand2 = SUM(ic_loct_inv.loct_onhand2)
2. The transaction quantity values are then used to update the summary inventory table (ic_summ_inv) as detailed for inventory movements above.

Inventory Quantities API - Error Messages

Listed below are all expected errors. These are output to the stored procedure message file and can be monitored through the return x_msg_count. This, in conjunction with the x_return_status can be used to monitor the success or failure of the procedure call.

Message Code	Narrative
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

Message Code	Narrative
IC_API_INVALID_LOT_STATUS	Invalid lot status for item number &ITEM_NO
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

Inventory Quantities API - Success Messages

If the Item Lot/Sublot conversion create is successful then the API returns a success message as follows:

Message Code	Narrative
IC_API_TRAN_POSTED	Inventory transaction posted for item &ITEM_NO lot &LOT_NO - &SUBLOT_NO

Translation of error messages is determined by the environment variable NLS_LANG. If no value can be determined, a default of 'US' is assumed. Message text is retrieved in accordance with the language setting.

7

Inventory Quantities API Wrapper

This topic provides business function and input structure for the Inventory Quantities API Wrapper. ASCII flat file layout, package and procedure names, special logic, and error messages are included. Sample inventory quantities API wrapper code is shown.

The following topics are covered:

- Inventory Quantities API Wrapper - Business Function
- Inventory Quantities API Wrapper - Input Structure
- Inventory Quantities API Wrapper - ASCII Flat File Layout
- Inventory Quantities API Wrapper - Package and Procedure Names
- Inventory Quantities API Wrapper - Special Logic
- Inventory Quantities API Wrapper - Error Messages
- Inventory Quantities API Wrapper - Code Example

Inventory Quantities API Wrapper - Business Function

This stored procedure is designed to operate in conjunction with the inventory quantities API. It may be required to be used in both synchronous (that is, on-line) and asynchronous (that is, batch) modes. When used in synchronous mode, the calling program (for example, an Oracle form) calls the API directly.

This topic is discusses using the API in asynchronous mode via a wrapper function. The source of data for the wrapper will come from:

- An ASCII flat file

This topic describes how the wrapper function should be called, and parameters that are required or optional.

Inventory Quantities API Wrapper - Input Structure

The API wrapper consists of a PL/SQL procedure and PL/SQL function both named 'Post'.

Inventory Quantities API Wrapper - ASCII Flat File Layout

As detailed previously, the ASCII flat file may be 'character delimited' (typically, but not necessarily, with a comma) or of fixed position format. The table below shows the order in which the data fields should appear and the maximum length of each data field.

Field Name	Type	Length	Required
transaction type	number	2	Y
item number	alphanumeric	32	Y
journal number	alphanumeric	32	N
from warehouse code	alphanumeric	4	Y
to warehouse code	alphanumeric	4	N
primary unit of measure	alphanumeric	4	Y
secondary unit of measure	alphanumeric	4	Y
lot number	alphanumeric	32	N
sublot number	alphanumeric	32	N
from location	alphanumeric	16	N

Field Name	Type	Length	Required
to location	alphanumeric	16	N
primary transaction quantity	number		N
secondary transaction quantity	number		N
QC grade	alphanumeric	4	N
lot status	alphanumeric	4	N
company code	alphanumeric	4	Y
organization code	alphanumeric	4	
transaction date	date (DDMMYYYY)	8	Y
reason code	alphanumeric	4	Y
user name	alphanumeric	100	N

Omitting an optional field is achieved by using consecutive delimiters.

Inventory Quantities API Wrapper - Package and Procedure Names

The Inventory Quantities API wrapper PL/SQL stored procedures code are held in the package:

- GMI_QUANTITY_WRP

The procedure or function to be called to execute this API wrapper is:

- Post

Inventory Quantities API Wrapper - Special Logic

Validation

Incorrectly formatted flat files are rejected.

The success or failure of the wrapper may be reported back to the calling function by means of the return value. This is hierarchically as follows:

- On initial entry to the wrapper the return status is set to success ('S').
- If for any record processed an expected error occurs and the return status is currently set to success then it is updated to expected error ('E').
- If for any record processed and unexpected error occurs, then the return status is set to unexpected error ('U').

Update Logic

Updates are only concerned with the processing of messages (errors and others) generated by the inventory quantities API.

Messages (success and error) are written to a flat file as designated by the p_ output_file parameter. Additionally a log file are written to the /tmp directory. This details start and completion times, data retrieved from the ASCII flat file and messages generated.

Inventory Quantities API Wrapper - Error Messages

Error messages generated by the inventory quantities API are written to the file 'p_ output_file'.

Errors listed below may be generated by the API wrapper. These messages are generally related to the handling of the ASCII flat input and output files. These messages are sent to the standard output device since it is inappropriate to attempt to send them to the files which themselves may be causing the erroneous condition. They are hard-coded in English.

Error Condition	Narrative
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

Inventory Quantities API Wrapper - Code Example

The PL/SQL code for this API wrapper is as follows:

```
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
CREATE OR REPLACE PACKAGE BODY GMI_QUANTITY_WRP AS
-- $Header: GMIPQTWB.pls 115.7 2000/08/10 15:02:24 hverddin gmigapib.pls $


--+=========================================================================+
--| PROCEDURE NAME                                                          |
--|    Post                                                                 |
--|                                                                         |
--| TYPE                                                                    |
--|    Public                                                               |
--|                                                                         |
--| USAGE                                                                   |
--|    Post an inventory transaction                                        |
--|                                                                         |
--| DESCRIPTION                                                             |
--|    This is a PL/SQL wrapper procedure to call the Post transaction      |
--|    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                                                                 |
--|                                                                         |
--+=========================================================================+
-- Api end of comments
PROCEDURE Post
( 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  :=Post( p_dir
      , p_input_file
      , p_output_file
      , p_delimiter
      );

End Post;

--+============================================================================+
--| FUNCTION NAME                                                              |
--|    Post                                                                    |
--|                                                                            |
--| TYPE                                                                       |
--|    Public                                                                  |
--|                                                                            |
--| USAGE                                                                      |
--|    Post an inventory transaction                                           |
--|                                                                            |
--| DESCRIPTION                                                                |
--|    This is a PL/SQL wrapper function to call the OPM                       |
--|    Inventory Quantities 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                                                                    |
--|    None                                                                    |
--|                                                                            |
--| HISTORY                                                                    |
--|                                                                            |
--+============================================================================+
-- Api end of comments
FUNCTION Post
( 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_loop_cnt           NUMBER  :=0;
l_dummy_cnt          NUMBER  :=0;
l_record_count       NUMBER  :=0;
l_data               VARCHAR2(2000);
trans_rec            GMIGAPI.qty_rec_typ;
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(200);
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);

l_ic_jrnl_mst_row  ic_jrnl_mst%ROWTYPE;
l_ic_adjs_jnl_row1 ic_adjs_jnl%ROWTYPE;
l_ic_adjs_jnl_row2 ic_adjs_jnl%ROWTYPE;
BEGIN

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

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');

-- Set the Wrapper file to be placed in the default working directory

l_log_dir  := p_dir;

--
-- Open The Wrapper File For Output And The Input File for Input.
--
dbms_output.put_line(l_log_name||' '||l_input_file||' '||l_log_dir||' '||l_p_
dir);
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
--

dbms_output.put_line('Start Processing');
UTL_FILE.PUT_LINE(l_log_handle, 'Process Started at '
|| to_char(SYSDATE,'DD-MON-YY HH: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');
dbms_output.put_line('Opened Log file: '||l_p_dir||l_output_file);

LOOP
l_record_count    :=l_record_count+1;
```

```
  BEGIN
  UTL_FILE.GET_LINE(l_infile_handle, l_line);
  dbms_output.put_line('LINE IS ' ||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 );
  trans_rec.trans_type      :=TO_NUMBER(Get_Field(l_line,l_delimiter,1));
  trans_rec.item_no         :=Get_Field(l_line,l_delimiter,2);
  trans_rec.journal_no      :=Get_Field(l_line,l_delimiter,3);
  trans_rec.from_whse_code  :=Get_Field(l_line,l_delimiter,4);
  trans_rec.to_whse_code    :=Get_Field(l_line,l_delimiter,5);
  trans_rec.item_um         :=Get_Field(l_line,l_delimiter,6);
  trans_rec.item_um2        :=Get_Field(l_line,l_delimiter,7);
  trans_rec.lot_no          :=Get_Field(l_line,l_delimiter,8);
  trans_rec.sublot_no       :=Get_Field(l_line,l_delimiter,9);
  trans_rec.from_location   :=Get_Field(l_line,l_delimiter,10);
  trans_rec.to_location     :=Get_Field(l_line,l_delimiter,11);
  trans_rec.trans_qty       :=TO_NUMBER(Get_Field(l_line,l_delimiter,12));
  trans_rec.trans_qty2      :=TO_NUMBER(Get_Field(l_line,l_delimiter,13));
  trans_rec.qc_grade        :=Get_Field(l_line,l_delimiter,14);
  trans_rec.lot_status      :=Get_Field(l_line,l_delimiter,15);
  trans_rec.co_code         :=Get_Field(l_line,l_delimiter,16);
  trans_rec.orgn_code       :=Get_Field(l_line,l_delimiter,17);
  IF Get_Field(l_line,l_delimiter,18)  IS NULL
  THEN
    trans_rec.trans_date    :=SYSDATE;
  ELSE
    trans_rec.trans_date    :=TO_DATE(Get_Field(l_line,l_delimiter,18)
     ,'DDMMYYYY');
  END IF;
  trans_rec.reason_code     :=Get_Field(l_line,l_delimiter,19);
  IF ((Get_Field(l_line,l_delimiter,20)) IS NULL)
  THEN
    trans_rec.user_name     :='OPM';
  ELSE
    trans_rec.user_name     :=Get_Field(l_line,l_delimiter,20);
  END IF;

UTL_FILE.PUT_LINE(l_log_handle,'trans type     = '||trans_rec.trans_type);
UTL_FILE.PUT_LINE(l_log_handle,'item no        = '||trans_rec.item_no);
UTL_FILE.PUT_LINE(l_log_handle,'journal no     = '||trans_rec.journal_no);
UTL_FILE.PUT_LINE(l_log_handle,'from_whse_code = '||
```

```
trans_rec.from_whse_code);
 UTL_FILE.PUT_LINE(l_log_handle,'to_whse_code   = '||
trans_rec.to_whse_code);
 UTL_FILE.PUT_LINE(l_log_handle,'item_um        = '||trans_rec.item_um);
 UTL_FILE.PUT_LINE(l_log_handle,'item_um2       = '||trans_rec.item_um2);
 UTL_FILE.PUT_LINE(l_log_handle,'lot no         = '||trans_rec.lot_no);
 UTL_FILE.PUT_LINE(l_log_handle,'sublot no      = '||trans_rec.sublot_no);
 UTL_FILE.PUT_LINE(l_log_handle,'from_location  = '||
trans_rec.from_location);
 UTL_FILE.PUT_LINE(l_log_handle,'to_location    = '||
trans_rec.to_location);
 UTL_FILE.PUT_LINE(l_log_handle,'trans_qty      = '||trans_rec.trans_qty);
 UTL_FILE.PUT_LINE(l_log_handle,'trans_qty2     = '||trans_rec.trans_qty2);
 UTL_FILE.PUT_LINE(l_log_handle,'qc_grade       = '||trans_rec.qc_grade);
 UTL_FILE.PUT_LINE(l_log_handle,'lot_status     = '||trans_rec.lot_status);
 UTL_FILE.PUT_LINE(l_log_handle,'co code        = '||trans_rec.co_code);
 UTL_FILE.PUT_LINE(l_log_handle,'orgn code      = '||trans_rec.orgn_code);
 UTL_FILE.PUT_LINE(l_log_handle,'trans_date     = '||trans_rec.trans_date);
 UTL_FILE.PUT_LINE(l_log_handle,'reason code    = '||trans_rec.reason_code);
 UTL_FILE.PUT_LINE(l_log_handle,'user name      = '||trans_rec.user_name );


 -- Allow Default Allocation Of User If NULL.
 IF trans_rec.user_name IS NULL THEN
    trans_rec.user_name :='OPM';
 END IF;

 GMIPAPI.Inventory_Posting
 ( 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_qty_rec  => trans_rec
 , x_ic_jrnl_mst_row => l_ic_jrnl_mst_row
 , x_ic_adjs_jnl_row1 => l_ic_adjs_jnl_row1
 , x_ic_adjs_jnl_row2 => l_ic_adjs_jnl_row2
 , 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);


  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 HH: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
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN UTL_FILE.INVALID_PATH THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN UTL_FILE.INVALID_MODE THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN UTL_FILE.INVALID_FILEHANDLE THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN UTL_FILE.WRITE_ERROR THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN UTL_FILE.READ_ERROR THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN UTL_FILE.INTERNAL_ERROR THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

WHEN OTHERS THEN
   UTL_FILE.FCLOSE_ALL;
RETURN l_return_status;

END Post;

--+============================================================================+
```

```
--| FUNCTION NAME                                                          |
--|    Get_Field                                                           |
--|                                                                        |
--| TYPE                                                                   |
--|    Public                                                              |
--|                                                                        |
--| USAGE                                                                  |
--|    Get value of field n from a delimited line of ASCII data            |
--|                                                                        |
--| DESCRIPTION                                                            |
--|    This utility function will return the value of a field from         |
--|    a delimited line of ASCII text                                      |
--|                                                                        |
--| PARAMETERS                                                             |
--|    p_line             IN VARCHAR2         - line of data               |
--|    p_delimiter        IN VARCHAR2         - Delimiter character        |
--|    p_field_no         IN NUMBER           - Field occurance to be      |
--|                                             returned                   |
--|                                                                        |
--| RETURNS                                                                |
--|    VARCHAR2                               - Value of field             |
--|                                                                        |
--| 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_QUANTITY_WRP;
/
commit;
exit;
```

8

Lot Create API

This topic provides the business function, technical overview and parameters for the Lot Create API. Also included are table and view usage, package and procedure names, special logic and error messages.

The following topics are covered:

- Lot Create API - Business Function
- Lot Create API - Technical Overview
- Lot Create API - Parameters for Create Lot
- Lot Create API - Table and View Usage
- Lot Create API - Package and Procedure Names
- Lot Create API - Special Logic
- Lot Create API - Error Messages
- Lot Create API - Code Example

Lot Create API - Business Function

This stored procedure is concerned with the following function within the Oracle Process Manufacturing (OPM) Inventory Management module:

- Create a lot for an inventory item

This topic describes how the stored procedure should be called, the parameters that are required (and optional) and the values that are returned to the calling program. This includes all error conditions that may arise.

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 allow Lot Update, or Delete.

Lot Create API - Technical Overview

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 that is described in this topic. The wrapper function passes the appropriate parameters into the stored procedure and is responsible for handling the return code from the procedure.

Lot Create API - Parameters for Create Lot

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  GMIGAPI.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 4, and last 3 parameters are standard across all of the API calls and are identical to the Create Item API where they are fully documented. If the lot creation is successful, 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 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 should be 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 SY$CPG_INSTALL flag are treated in the same way.

The p_lot_rec parameter is used to pass the lot-specific data required to create a lot for an inventory item. It is described below. Please refer to the Lot Create API Wrapper topic for an example of how to populate this parameter and call the stored procedure.

Field/Column	Type	Length	Default	Req'd	Validation
item_no	varchar2	32		Y	Must exist ic_item_mst Must not be deleted Must be active Must be lot-controlled
lot_no	varchar2	32		Y	Non-blank
sublot_no	varchar2	32		N	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	varchar2	40		N	
qc_grade	varchar2	4	ic_item_mst.qc_grade	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	varchar2	4	ic_item_mst.expaction_code	N	Must exist on qc_actn_mst if non-blank

Field/Column	Type	Length	Default	Req'd	Validation
expaction_date	date		IF ic_item_ mst.grade_ctl = 1AND ic_item_ mst.expaction_ interval > 0 THEN expire_ date + ic_item_ mst.expaction_ interval ELSE 31-Dec-2010 00:00:00 (SY$MAX_ DATE)	N	Must not be less than lot_created date Must not be less than expire_date
lot_created	date		System Date	N	
expire_date	date		IF ic_item_ mst.grade_ctl = 1AND ic_item_ mst.shelf_life > 0 THEN lot_ created + ic_ item_mst.shelf_ life ELSE 31-Dec-2010 00:00:00 (SY$MAX_ DATE)	N	Must not be less than lot_created date
retest_date	date		IF ic_item_ mst.grade_ctl = 1AND 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 not be less than lot_created date
strength	number		100	N	Must be zero or positive value

Field/Column	Type	Length	Default	Req'd	Validation
inactive_ind	number	5	0	N	Must be 0 or 1
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	varchar2	32		N	Must exist on po_vend_mst if non-blank
vendor_lot_no	varchar2	32		N	
ic_matr_date	date		lot_created + ic_item_mst.ic_ matr_days	N	
ic_hold_date	date		lot_created + ic_item_mst.ic_ hold_days	N	
attribute1	varchar2	240		N	Descriptive flexfield segment
attribute2	varchar2	240		N	Descriptive flexfield segment
attribute3	varchar2	240		N	Descriptive flexfield segment
attribute4	varchar2	240		N	Descriptive flexfield segment
attribute5	varchar2	240		N	Descriptive flexfield segment
attribute6	varchar2	240		N	Descriptive flexfield segment
attribute7	varchar2	240		N	Descriptive flexfield segment
attribute8	varchar2	240		N	Descriptive flexfield segment

Field/Column	Type	Length	Default	Req'd	Validation
attribute9	varchar2	240		N	Descriptive flexfield segment
attribute10	varchar2	240		N	Descriptive flexfield segment
attribute11	varchar2	240		N	Descriptive flexfield segment
attribute12	varchar2	240		N	Descriptive flexfield segment
attribute13	varchar2	240		N	Descriptive flexfield segment
attribute14	varchar2	240		N	Descriptive flexfield segment
attribute15	varchar2	240		N	Descriptive flexfield segment
attribute16	varchar2	240		N	Descriptive flexfield segment
attribute17	varchar2	240		N	Descriptive flexfield segment
attribute18	varchar2	240		N	Descriptive flexfield segment
attribute19	varchar2	240		N	Descriptive flexfield segment
attribute20	varchar2	240		N	Descriptive flexfield segment
attribute21	varchar2	240		N	Descriptive flexfield segment
attribute22	varchar2	240		N	Descriptive flexfield segment
attribute23	varchar2	240		N	Descriptive flexfield segment
attribute24	varchar2	240		N	Descriptive flexfield segment
attribute25	varchar2	240		N	Descriptive flexfield segment

Field/Column	Type	Length	Default	Req'd	Validation
attribute26	varchar2	240		N	Descriptive flexfield segment
attribute27	varchar2	240		N	Descriptive flexfield segment
attribute28	varchar2	240		N	Descriptive flexfield segment
attribute29	varchar2	240		N	Descriptive flexfield segment
attribute30	varchar2	240		N	Descriptive flexfield segment
attribute_ category	varchar2	30		N	Descriptive flexfield structure definition
user_name	varchar2	100	'OPM'	N	Ignored but retained for backward compatibility

Lot Create API - Table and View Usage

The following OPM tables are referenced by the Lot Create API. The appropriate entries in these tables must exist and be non-delete marked on the database for validation usage through the Item Create stored procedure.

Table Name	Select	Insert	Update	Delete	Base Table
ic_item_mst	X				
sy_type_mst	X				
qc_grad_mst	X				
po_vend_mst	X				
qc_actn_mst	X				
ic_item_cpg	X				
ic_lots_mst	X	X			
ic_lots_cpg		X			
ic_lots_sts	X				
sy_oper_mst	X				

The Item Master table IC_ITEM_MST holds all attributes of the item.

Lot controlled and QC attributes are prompted for in the maintenance form once the appropriate indicators are set. These details are also held against the item.

Whether an item is defined as lot controlled or not the system always generates a DEFAULT lot against the item on IC_LOTS_MST.

All lots hold additional attributes for Consumer Packaged Goods (CPG) specific processing. These are held against IC_LOTS_CPG.

When an item is identified as dual UOM and the alternate UOM is of a different type (for example, an item is defined in type WEIGHT with dual UOM in VOLUME) saving the item master prompts you for the conversion factor between WEIGHT and VOLUME for the item. This mechanism and structure supports transactional processing of the item in any supported UOM. This is handled outside of the Item Create procedure through an additional function.

Additional text can be maintained against the item to hold free format information on IC_TEXT_TBL. This is handled outside of the Item Create procedure through an additional function.

Lot Create API - Package and Procedure Names

The Lot Create API PL/SQL stored procedure code is held in the following package:

- GMIPAPI

The stored procedure which is called to create a new item lot is:

- Create_Lot

Please refer to the Lot Create API Wrapper topic for an example of how the previous procedure is executed.

Lot Create API - Special Logic

Update Logic

When all the validation checks have been performed and no errors found, a new lot is created within the database. The following steps are followed:

1. If p_lot_rec.lot_no = GMI:Default Lot then
   - set lot_id = 0
   - otherwise select the next lot_id from sequence gem5_lot_id_s.
2. Build row in ic_lots_mst as follows:
   - All matching column names in PL/SQL record p_item_rec are transferred directly into the corresponding ic_lots_mst columns.
   - The following columns on ic_lots_mst require further explanation:

Column Name	Value
item_id	ic_item_mst.item_id
lot_id	surrogate key as obtained above
shipvend_id	po_vend_mst.vend_id
date_added	system date

Column Name	Value
date_modified	System date
added_by	p_lot_rec.op_code
modified_by	p_lot_rec.op_code
trans_cnt	1
delete_mark	0
text_code	0

3. If SY$CPG_INSTALL_FLAG=1, build row in ic_lots_cpg as follows:

Column Name	Value
item_id	ic_item_mst.item_id
lot_id	Surrogate key as obtained above
ic_matr_date	Calculated as System date + p_lot_rec.ic_matr_days
ic_hold_date	Calculated as System date + p_lot_rec.ic_hold_days
created_by	Standard who column updated with FND_ USER.USER_ID based on supplied user_name
creation_date	System date
last_updated_by	Standard who column updated with FND_ USER.USER_ID based on supplied user_name
last_update_date	System date
last_update_login	login id

Lot Create API - Error Messages

Listed below are all expected errors. These are output to the stored procedure message file and can be monitored through the return x_msg_count. This, in conjunction with the x_return_status can be used to monitor the success or failure of the procedure call.

Message Code	Narrative
IC_API_INVALID_ITEM	Invalid item &ITEM
IC_API_ITEM_NOT_LOT_CTL	Item &ITEM is not lot controlled
IC_API_SUBLOT_NOT_REQD	Sub-lot &SUBLOT supplied for item &ITEM, lot &LOT which is not sub-lot controlled
IC_API_LOT_ALREADY_EXISTS	Lot &LOT - &SUBLOT for item number &ITEM already exists
IC_API_INVALID_LOT_QC_GRADE	Invalid QC grade for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_EXPACTION_CODE	Invalid expiry action for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_EXPACTION_DATE_API	Invalid expiry action date for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_EXPIRE_DATE	Invalid expire date for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_RETEST_DATE	Invalid retest date for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_STRENGTH	Invalid strength for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_INACTIVE_IND	Invalid inactive indicator for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_ORIGINATION_ TYPE	Invalid origination type for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_SHIPVENDOR_NO	Invalid ship vendor for item &ITEM lot &LOT - &SUBLOT
IC_API_INVALID_LOT_MATR_DAYS	Invalid maturity days for item &ITEM lot &LOT - &SUBLOT

Message Code	Narrative
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

9

Lot Create API Wrapper

This topic provides the business function and ASCII flat file layout for the lot create API wrapper. Also included are package and procedure names, special logic and error messages. A code example is included.

The following topics are covered:

- Lot Create API Wrapper - Business Function
- Lot Create API Wrapper - ASCII Flat File Layout
- Lot Create API Wrapper - Package and Procedure Names
- Lot Create API Wrapper - Special Logic
- Lot Create API Wrapper - Error Messages
- Lot Create API Wrapper - Code Example

Lot Create API Wrapper - Business Function

This stored procedure is designed to operate in conjunction with the Lot Create API. This is used to create an inventory item lot in OPM. It may be required to be used in both synchronous (that is, on-line) and asynchronous (that is, batch) modes. When used in synchronous mode, the calling program (for example, an Oracle form) calls the API directly. Using the API in synchronous mode is not considered in this topic.

This topic is concerned with using the API in asynchronous mode using a wrapper function. The source of data for the wrapper comes from:

- An ASCII flat file

This topic describes how the wrapper function should be called and the parameters that are required (and optional).

Lot Create API Wrapper - ASCII Flat File Layout

The ASCII flat file may be 'character delimited' (typically, but not necessarily, with a comma) or of fixed position format. The table below shows the order in which the data fields should appear (for delimited files) and the length of each data field (for fixed position format files).

Field/Column	Type	Length	Required
lot number	alphanumeric	32	Y
sublot number	alphanumeric	32	N
lot description	alphanumeric	40	N
p_init_msg_list determines if the error message stack should be purged. Normally this parameter is omitted and defaulted as above (QC) grade	alphanumeric	4	N
expiry action code	alphanumeric	4	N
expiry action date	date (DDMMYYYY)	8	N
lot created date	date (DDMMYYYY)	8	N
lot expire date	date (DDMMYYYY)	8	N

Field/Column	Type	Length	Required
retest date	date (DDMMYYYY)	8	N
strength	number	16	N
inactive indicator	number	5	N
origination type	number	5	N
user class1	alphanumeric	16	N
user class2	alphanumeric	16	N
user class3	alphanumeric	16	N
user class4	alphanumeric	16	N
user class5	alphanumeric	16	N
user class6	alphanumeric	16	N
ship vendor number	alphanumeric	32	N
vendors lot number	alphanumeric	32	N
lot maturity date	date (DDMMYYYY)	8	N
lot hold release date	date (DDMMYYYY)	8	N
operator code	alphanumeric	4	Ignored

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

Lot Create API Wrapper - Package and Procedure Names

The Lot Create API wrapper PL/SQL stored procedures code is held in the package:

- GMI_LOTS_WRP

The procedure or function to be called to execute this API wrapper is:

- Create_Lot

Lot Create API Wrapper - Special Logic

Validation

Incorrectly formatted flat files are rejected.

The success or failure of the wrapper may be reported back to the calling function by means of the return value. This is hierarchically as follows:

- On initial entry to the wrapper the return status will be set to success ('S').
- If for any record processed an expected error occurs and the return status is currently set to success then it will be updated to expected error ('E').
- If for any record processed and unexpected error occurs, then the return status is set to unexpected error ('U').

Update Logic

Updates are only be concerned with the processing of messages (errors and others) generated by the Lot Create API.

Messages (success and error) are written to a flat file as designated by the p_ output_file parameter. Additionally a log file are written to the /tmp directory. This details start and completion times, data retrieved from the ASCII flat file and messages generated.

Lot Create API Wrapper - Error Messages

Error messages generated by the Lot Create API are written to the file ‘p_output_file’.

Errors listed below may be generated by the API wrapper. These messages are generally related to the handling of the ASCII flat input and output files. These messages are sent to the standard output device as it is inappropriate to attempt to send them to the files which themselves may be causing the erroneous condition. They are hard-coded in English.

Error Condition	Narrative
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

Lot Create API Wrapper - Code Example

The actual PL/SQL code for this API is listed below.

```
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
CREATE OR REPLACE PACKAGE BODY GMI_LOTS_WRP AS
-- $Header: GMIPLOWB.pls 115.6 2000/10/02 19:06:28 pschofie gmigapib.pls $
--+=========================================================================+
--| PROCEDURE NAME                                                          |
--|    Create_Lot                                                           |
--|                                                                         |
--| TYPE                                                                    |
--|    Public                                                               |
--|                                                                         |
--| USAGE                                                                   |
--|    Create an Item Lot                                                   |
--|                                                                         |
--| DESCRIPTION                                                             |
--|    This is a PL/SQL wrapper procedure to call the Create_Lot            |
--|    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                                                                 |
--|                                                                         |
--+=========================================================================+
-- Api end of comments
PROCEDURE Create_Lot
( 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_Lot( p_dir
     , p_input_file
     , p_output_file
     , p_delimiter
     );

End Create_Lot;

--+============================================================================+
--| FUNCTION NAME                                                              |
--|    Create_Lot                                                              |
--|                                                                            |
--| TYPE                                                                       |
--|    Public                                                                  |
--|                                                                            |
--| USAGE                                                                      |
--|    Create a Lot/Sublot                                                     |
--|                                                                            |
--| DESCRIPTION                                                                |
--|    This is a PL/SQL wrapper function to call the OPM Lot/Sublot Create     |
--|    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_Lot
( 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_loop_cnt          NUMBER  :=0;
l_dummy_cnt         NUMBER  :=0;
l_record_count      NUMBER  :=0;
l_data              VARCHAR2(2000);
lot_rec             GMIGAPI.lot_rec_typ;
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(200);
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);
l_ic_lots_mst_row   ic_lots_mst%ROWTYPE;
l_ic_lots_cpg_row   ic_lots_cpg%ROWTYPE;
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
--

dbms_output.put_line('Start Processing');
UTL_FILE.PUT_LINE(l_log_handle, 'Process Started at '
|| to_char(SYSDATE,'DD-MON-YY HH: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 );

  lot_rec.item_no         :=Get_Field(l_line,l_delimiter,1);
  lot_rec.lot_no          :=Get_Field(l_line,l_delimiter,2);
  lot_rec.sublot_no       :=Get_Field(l_line,l_delimiter,3);
  lot_rec.lot_desc        :=Get_Field(l_line,l_delimiter,4);
  lot_rec.qc_grade        :=Get_Field(l_line,l_delimiter,5);
  lot_rec.expaction_code  :=Get_Field(l_line,l_delimiter,6);
  IF (Get_Field(l_line,l_delimiter,7) IS NULL)
  THEN
    lot_rec.expaction_date :=GMA_GLOBAL_GRP.SY$MAX_DATE;
  ELSE
    lot_rec.expaction_date :=TO_DATE(
    Get_Field(l_line,l_delimiter,7),'DDMMYYYY');
  END IF;
  IF (Get_Field(l_line,l_delimiter,8) IS NULL)
  THEN
    lot_rec.lot_created   :=SYSDATE;
  ELSE
    lot_rec.lot_created   :=TO_DATE(
    Get_Field(l_line,l_delimiter,8),'DDMMYYYY');
  END IF;
  IF (Get_Field(l_line,l_delimiter,9) IS NULL)
  THEN
    lot_rec.expire_date   :=GMA_GLOBAL_GRP.SY$MAX_DATE;
  ELSE
    lot_rec.expire_date   :=TO_DATE(
    Get_Field(l_line,l_delimiter,9),'DDMMYYYY');
  END IF;
  IF (Get_Field(l_line,l_delimiter,10) IS NULL)
  THEN
    lot_rec.retest_date   :=GMA_GLOBAL_GRP.SY$MAX_DATE;
  ELSE
    lot_rec.retest_date   :=TO_DATE(
    Get_Field(l_line,l_delimiter,10),'DDMMYYYY');
  END IF;
  IF (Get_Field(l_line,l_delimiter,11) IS NULL)
  THEN
    lot_rec.strength      :=100;
  ELSE
    lot_rec.strength      :=TO_NUMBER(Get_Field(l_line,l_delimiter,11));
  END IF;
  IF (Get_Field(l_line,l_delimiter,12) IS NULL)
```

```
THEN
  lot_rec.origination_type     :=0;
ELSE
  lot_rec.origination_type     :=TO_NUMBER(Get_Field(l_line,l_delimiter,12));
END IF;
lot_rec.shipvendor_no   :=Get_Field(l_line,l_delimiter,13);
lot_rec.vendor_lot_no   :=Get_Field(l_line,l_delimiter,14);
IF (Get_Field(l_line,l_delimiter,15) IS NULL)
THEN
  lot_rec.ic_matr_date   :=GMA_GLOBAL_GRP.SY$MAX_DATE;
ELSE
  lot_rec.ic_matr_date   :=TO_DATE(
   Get_Field(l_line,l_delimiter,15),'DDMMYYYY');
END IF;
IF (Get_Field(l_line,l_delimiter,16) IS NULL)
THEN
  lot_rec.ic_hold_date   :=GMA_GLOBAL_GRP.SY$MAX_DATE;
ELSE
  lot_rec.ic_hold_date   :=TO_DATE(
   Get_Field(l_line,l_delimiter,16),'DDMMYYYY');
END IF;
IF (Get_Field(l_line,l_delimiter,17) IS NULL)
THEN
  lot_rec.user_name       :='OPM';
ELSE
  lot_rec.user_name       :=Get_Field(l_line,l_delimiter,17);
END IF;
lot_rec.attribute1      :=Get_Field(l_line,l_delimiter,18);
lot_rec.attribute2      :=Get_Field(l_line,l_delimiter,19);
lot_rec.attribute3      :=Get_Field(l_line,l_delimiter,20);
lot_rec.attribute4      :=Get_Field(l_line,l_delimiter,21);
lot_rec.attribute5      :=Get_Field(l_line,l_delimiter,22);
lot_rec.attribute6      :=Get_Field(l_line,l_delimiter,23);
lot_rec.attribute7      :=Get_Field(l_line,l_delimiter,24);
lot_rec.attribute8      :=Get_Field(l_line,l_delimiter,25);
lot_rec.attribute9      :=Get_Field(l_line,l_delimiter,26);
lot_rec.attribute10     :=Get_Field(l_line,l_delimiter,27);
lot_rec.attribute11     :=Get_Field(l_line,l_delimiter,28);
lot_rec.attribute12     :=Get_Field(l_line,l_delimiter,29);
lot_rec.attribute13     :=Get_Field(l_line,l_delimiter,30);
lot_rec.attribute14     :=Get_Field(l_line,l_delimiter,31);
lot_rec.attribute15     :=Get_Field(l_line,l_delimiter,32);
lot_rec.attribute16     :=Get_Field(l_line,l_delimiter,33);
lot_rec.attribute17     :=Get_Field(l_line,l_delimiter,34);
lot_rec.attribute18     :=Get_Field(l_line,l_delimiter,35);
```

```
  lot_rec.attribute19     :=Get_Field(l_line,l_delimiter,36);
  lot_rec.attribute20     :=Get_Field(l_line,l_delimiter,37);
  lot_rec.attribute21     :=Get_Field(l_line,l_delimiter,38);
  lot_rec.attribute22     :=Get_Field(l_line,l_delimiter,39);
  lot_rec.attribute23     :=Get_Field(l_line,l_delimiter,40);
  lot_rec.attribute24     :=Get_Field(l_line,l_delimiter,41);
  lot_rec.attribute25     :=Get_Field(l_line,l_delimiter,42);
  lot_rec.attribute26     :=Get_Field(l_line,l_delimiter,43);
  lot_rec.attribute27     :=Get_Field(l_line,l_delimiter,44);
  lot_rec.attribute28     :=Get_Field(l_line,l_delimiter,45);
  lot_rec.attribute29     :=Get_Field(l_line,l_delimiter,46);
  lot_rec.attribute30     :=Get_Field(l_line,l_delimiter,47);
  lot_rec.attribute_category  :=Get_Field(l_line,l_delimiter,48);

UTL_FILE.PUT_LINE(l_log_handle,'item no        = '||lot_rec.item_no);
UTL_FILE.PUT_LINE(l_log_handle,'lot_no         = '||lot_rec.lot_no);
UTL_FILE.PUT_LINE(l_log_handle,'sublot_no      = '||lot_rec.sublot_no);
UTL_FILE.PUT_LINE(l_log_handle,'lot_desc       = '||lot_rec.lot_desc);
UTL_FILE.PUT_LINE(l_log_handle,'qc_grade       = '||lot_rec.qc_grade);
UTL_FILE.PUT_LINE(l_log_handle,'expaction_code = '||
lot_rec.expaction_code);
UTL_FILE.PUT_LINE(l_log_handle,'expaction_date = '||
lot_rec.expaction_date);
UTL_FILE.PUT_LINE(l_log_handle,'lot_created    = '||
lot_rec.lot_created);
UTL_FILE.PUT_LINE(l_log_handle,'expire_date    = '||
lot_rec.expire_date);
UTL_FILE.PUT_LINE(l_log_handle,'retest_date    = '||
lot_rec.retest_date);
UTL_FILE.PUT_LINE(l_log_handle,'strength       = '||lot_rec.strength);
UTL_FILE.PUT_LINE(l_log_handle,'origination_type= '||
lot_rec.origination_type);
UTL_FILE.PUT_LINE(l_log_handle,'shipvendor_no  = '||lot_rec.shipvendor_no);
UTL_FILE.PUT_LINE(l_log_handle,'vendor_lot_no  = '||lot_rec.vendor_lot_no);
UTL_FILE.PUT_LINE(l_log_handle,'ic_matr_date   = '||lot_rec.ic_matr_date);
UTL_FILE.PUT_LINE(l_log_handle,'ic_hold_date   = '||lot_rec.ic_hold_date);
UTL_FILE.PUT_LINE(l_log_handle,'user name        = '||lot_rec.user_name );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute1    = '||
lot_rec.attribute1 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute2    = '||
lot_rec.attribute2 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute3    = '||
lot_rec.attribute3 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute4    = '||
lot_rec.attribute4 );
```

```
UTL_FILE.PUT_LINE(l_log_handle,'Attribute5    = '||
lot_rec.attribute5 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute6    = '||
lot_rec.attribute6 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute7    = '||
lot_rec.attribute7 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute8    = '||
lot_rec.attribute8 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute9    = '||
lot_rec.attribute9 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute10    = '||
lot_rec.attribute10 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute11    = '||
lot_rec.attribute11 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute12    = '||
lot_rec.attribute12 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute13    = '||
lot_rec.attribute13 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute14    = '||
lot_rec.attribute14 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute15    = '||
lot_rec.attribute15 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute16    = '||
lot_rec.attribute16 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute17    = '||
lot_rec.attribute17 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute18    = '||
lot_rec.attribute18 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute19    = '||
lot_rec.attribute19 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute20    = '||
lot_rec.attribute20 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute21    = '||
lot_rec.attribute21 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute22    = '||
lot_rec.attribute22 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute23    = '||
lot_rec.attribute23 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute24    = '||
lot_rec.attribute24 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute25    = '||
lot_rec.attribute25 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute26    = '||
lot_rec.attribute26 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute27    = '||
```

```
lot_rec.attribute27 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute28    = '||
lot_rec.attribute28 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute29    = '||
lot_rec.attribute29 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute30    = '||
lot_rec.attribute30 );
UTL_FILE.PUT_LINE(l_log_handle,'Attribute_Category = '||
lot_rec.attribute_category );

GMIPAPI.Create_Lot
( 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_lot_rec        =>lot_rec
, x_ic_lots_mst_row => l_ic_lots_mst_row
, x_ic_lots_cpg_row => l_ic_lots_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 HH: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;

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

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

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

WHEN UTL_FILE.WRITE_ERROR THEN
   dbms_output.put_line('Invalid Write Error   '|| l_global_file);
   UTL_FILE.FCLOSE_ALL;

WHEN UTL_FILE.READ_ERROR THEN
   dbms_output.put_line('Invalid Read  Error   '|| l_global_file);
   UTL_FILE.FCLOSE_ALL;

WHEN UTL_FILE.INTERNAL_ERROR THEN
   dbms_output.put_line('Internal Error');
   UTL_FILE.FCLOSE_ALL;

WHEN OTHERS THEN
   dbms_output.put_line('Other Error');
   UTL_FILE.FCLOSE_ALL;

END Create_Lot;

--+=====================================================================+
--| FUNCTION NAME                                                       |
--|    Get_Field                                                        |
--|                                                                     |
--| TYPE                                                                |
--|    Public                                                           |
--|                                                                     |
--| USAGE                                                               |
--|    Get value of field n from a delimited line of ASCII data         |
--|                                                                     |
--| DESCRIPTION                                                         |
--|    This utility function will return the value of a field from      |
--|    a delimited line of ASCII text                                   |
--|                                                                     |
--| PARAMETERS                                                          |
--|    p_line             IN VARCHAR2         - line of data            |
--|    p_delimiter        IN VARCHAR2         - Delimiter character     |
--|    p_field_no         IN NUMBER           - Field occurance to be   |
```

```
--|                                                 returned                  |
--|                                                                           |
--| RETURNS                                                                   |
--|    VARCHAR2                                    - Value of field           |
--|                                                                           |
--| 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_LOTS_WRP;
/
commit;
exit;
```

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X