

Conversions, Interfaces and Oracle Applications: Data Movement throughout the Oracle Applications Life Cycle

DataMirror[®]
The experience of now.[™]

Contents

Contents.....	1
Management Summary	1
The Challenge.....	1
What do we convert?	1
Figuring out how to get the data into the Applications	2
Entering data using Applications screens.....	2
Entering data using the Open Systems Interface.....	3
Entering data into the tables directly	3
Cleaning the data.....	4
Determining what interfaces are needed.....	4
Designing and Developing the Interface Process.....	5
Testing the interface.....	6
Moving the interface into production	6
Producing Interfaces Faster.....	7
About Constellar [®] Hub	8

Management Summary

The challenges of implementing conversions and interfaces in an Oracle Application implementation are many. If the Development Team determines a strategy and process to approach these challenges, however, the task can become manageable. In order to meet project constraints, such as schedule, money, and headcount, a tool is virtually always required to develop the range and quantity of conversions and interfaces needed. The use of a tool also reduces the cost of ownership, as well.

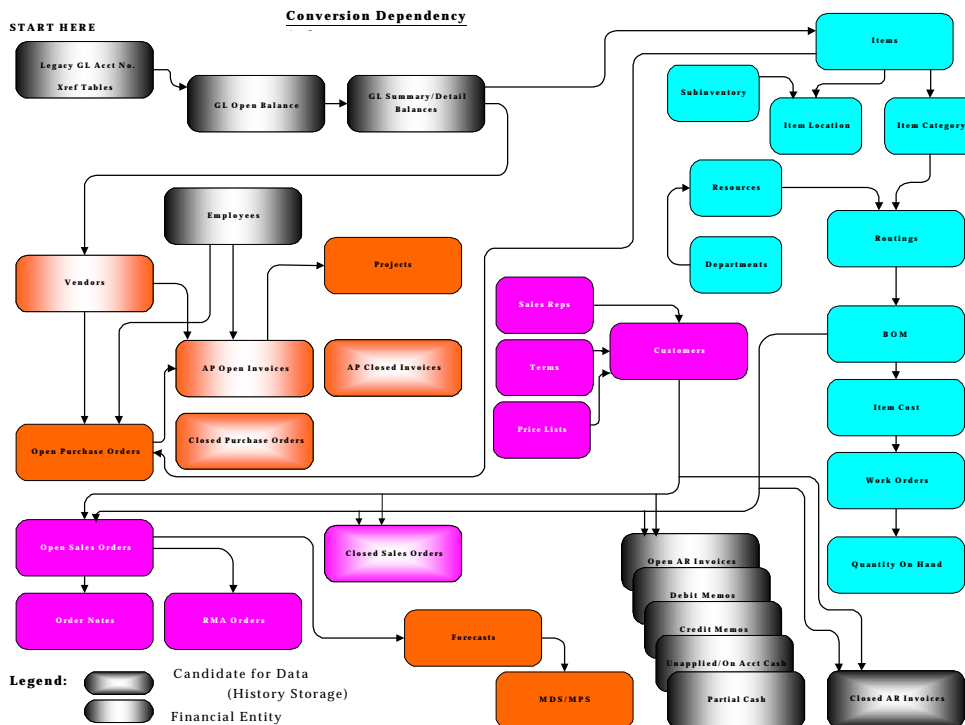
The Challenge

The Oracle Applications Implementation Team faces many challenges regarding data migration when implementing Oracle Apps. They include:

- What do I convert?
- Figuring out how to get the data into the Applications
- Getting the data to convert
- Cleaning the data
- Determining what interfaces are needed
- Designing and developing the interface process
- Testing the interface
- Moving the interface into production
- Producing Interfaces Faster

What do we convert?

In order to start a new instance of Oracle Applications, a number of sets of data must be provided first. The diagram below illustrates the data objects that must be populated, depending on which modules are being implemented, and the order they must be implemented (in Oracle Applications release 10).



Figuring out how to get the data into the Applications

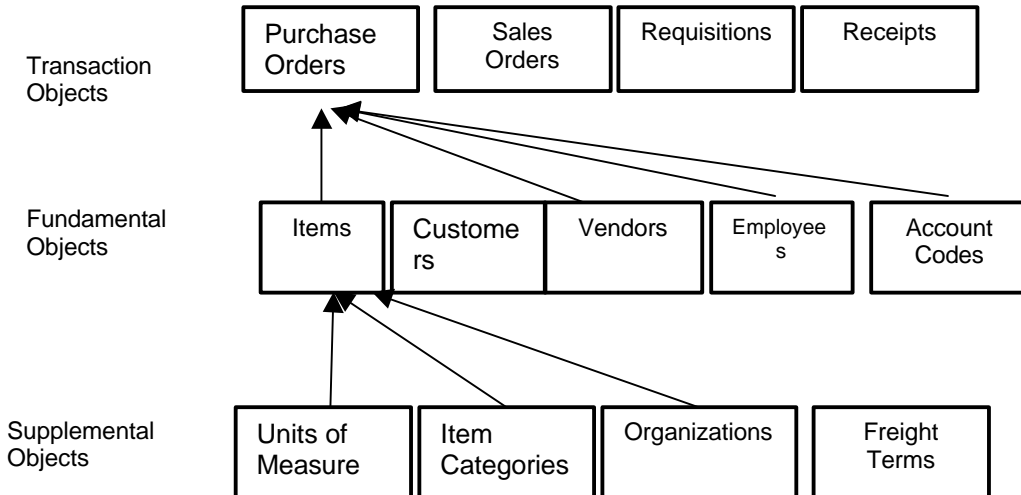
There are three alternatives to getting data into Oracle Applications :

1. The data can be entered using the Applications screens
2. The data can be entered using Oracle's Open Systems Interfaces
3. The data can be stored in the database tables directly

Entering data using Applications screens

Entering data into the screens is the safest, but most time consuming method for getting data into the Apps. It is also prone to quality problems, if people are manually typing data each time an instance of the apps is populated. The potential for inconsistency through human error is very high.

There are ways of using Excel Macros to take data from an Excel spreadsheet and stuff them into the Oracle Applications screens. The screens that can be stuffed are quite few, however. This technique works best for elemental items, like freight terms, employees, and simple lists of values.



The objects in an Oracle Applications database can be generalized as belonging in one of three major groups: Supplemental Objects, Fundamental Objects, and Transaction Objects. A given higher object consists of foreign key references to subordinate objects. In determining the order that conversions must be executed, it must be remembered to start at the bottom and work up the hierarchy.

In the (simplified) hierarchical diagram above, a purchase order consists of a combination of fundamental objects, namely Items, purchased from Vendors, by a certain Employee, and charged to a particular Account Code. A given item consists of values from a Units of Measure table, optional categories, organizations, and so on. Screen stuffers work best at the Supplemental object level, because there are no foreign keys. Screen stuffers are inappropriate with most Fundamental Objects and nearly too difficult to implement with Transaction Objects. Unfortunately for the user, Oracle Corporation does not provide Open Systems Interfaces for the Supplemental Objects and nearly half of the Fundamental objects.

Entering data using the Open Systems Interface

It is a good idea to use Oracle's Open System Interfaces (OSI) whenever possible. It is also a good idea to write one piece of software, however, that can function as both a conversion and an interface. Some of the OSIs require a bit of work to figure out. This is because the behavior of the OSI varies according to the unique setups entered by the user. Therefore, the behavior of the Open Item Interface is likely to be different from one Oracle Applications customer to another.

Entering data into the tables directly

This is usually not advised. It carries with it the tremendous risk of missing a referential integrity nuance or the validation of some field. If the database is populated with technically bad data, returning the system to a functional state may not be possible. This is one reason why Oracle does not advise customers updated database tables directly.

Having stated that, there will be occasions where you must update the tables directly. The best course of action is to find a consultant who has experience with the particular business objects you wish to load. Also, remember it is easier to figure out the Supplemental Objects than the Fundamental or Transactional Objects. Do not contemplate updating Fundamental or Transactional Object tables directly because the referential integrity and data validation is not easily discovered or replicated.

Cleaning the data

The task of converting questionable data into reliable, clean data can be a daunting one. It is frequently best to have a separate project that is responsible for producing clean data for the implementation. The data should be divided into groups, according to process. A specific individual should be responsible for leading the cleanup initiative. Usually, a Information Systems person is not the best choice for this role, since the data is best understood by users. Many IT departments say, regarding data, that it is IT's responsibility for determining the quality of the storage place and the responsibility of the user to determine the quality of what gets stored. In other words, it is the user's data stored in IT's structures.

There are two kinds of data quality that is relevant to Conversions and Interfaces :
Technical quality and Business quality.

The technical quality of the data encompasses :

1. What is the quality of the referential integrity? In other words, if a file of requisitions contains a buyer code, the record is considered bad if the buyer code is not in the list of valid buyers. If a requisition also contains item numbers, are there any items in the file which are not found in the item master? Are there units of measure not found in the unit of measure table?
2. What is the consistency of the data? This refers to whether the arrangement of the fields within the data is consistent and whether the data within a given field is of the same data type.
3. For files with self-referencing data, where the child of a particular record appears as a parent in the file, are there any recursive relationships? For example, in a BOM, is there a part that has itself as a subordinate parent? In other words, if part A contains part B, and part B contains part A, a recursive relationship exists. Files of self-referencing data should not have recursive relationships.

The business quality of the file relates to how well the data in the file adheres to a set of business rules unique to each business. The most difficult part of assessing business quality is gaining a consensus on what those rules should be. For example, a business might decide that a purchase order is considered obsolete if all line items have been received and the last receipt of material is more than two years old, it is to be rejected from the file. A business may decide that sales history for a particular discontinued product line are to be rejected. The business quality rules may also require specific conversions to take place. For example, if the business is changing the chart of accounts in implementing Oracle Applications, then the cross reference table should be part of the business rules.

Cross reference tables should ideally permit two-way conversion – old values to new and new values back to old. This is especially important when data must be sent back to legacy systems. The key values must be cross-referenced back to a previous standard.

When validating data quality, be sure that the reports used are accurate. Many a long night has been spent looking for quality problems, only to find out that the report was filtering out records, had an internal conversion or cross-reference, or otherwise misled the unwary analyst.

Determining what interfaces are needed

Early in a project, it isn't always clear what interfaces will be required. The best place to start is by obtaining a Current State Diagram. This diagram shows the applications and interfaces between them

One must know what the goal of the implementation is. Therefore, an End State Diagram must be agreed upon and drawn. Then, the question of process of moving from the Current State to the End State must be raised: How do we get from the Current State Diagram to the End State Diagram? If it is to be done in phases, then which applications will remain untouched at a particular point in time and which applications will be phased out at that same point in time. This will result in the production of Intermediate State Diagrams. From that point, it is a simple matter to document high level data about the flows of information between the applications in each Intermediate State Diagram. For each flow, an interface will be required. For each interface, there are things which must be determined:

Interface Category - Purchasing, Finance, Manufacturing, etc.

Date Required - When is this conversion or interface needed?

Frequency - One-time initial load, scheduled (when?), ad hoc

Purpose of Flow - Description of business purpose fulfilled by flow

Remote Host, Operating System - Where does it come from?

Remote Software System Name

Oracle Apps Module Name – INV, MFG, GL, OE, etc.

Direction of Flow to Oracle Apps - Incoming or outgoing

Transport Mechanism - FTP, NDM, LU6.2, RPC, etc

Origination Scheme - How is the flow originated?

Reception Scheme - How is the flow processed?

Confirmation Required - How do we confirm that it was done correctly?

Data Source Field Formats - Field name, type, size, purpose

Data Target Field Formats - Field name, type, size, purpose

Known Conversions – Converting account codes, for example

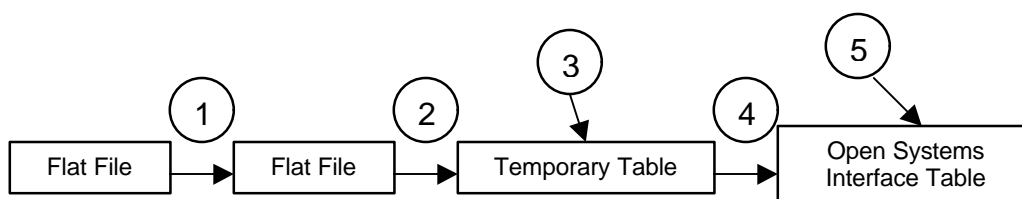
Known Business Rules - All POs older than two years are to be ignored, for example

It is important to document these interfaces carefully. Since it is less expensive to implement changes early in the development lifecycle, the sooner attention and scrutiny is focused on the interfaces, the fewer the number of errors are likely to appear.

Designing and Developing the Interface Process

The design of an interface should permit ease of troubleshooting and maintenance. It is usually best to divide the work into distinct modules, each of which can be more easily tested. The modular design also makes changes easier and faster to implement. Figure X contains a general flow for an inbound interface. All steps in this interface would be controlled by an executive process, like a UNIX shell script running under CRON, an AppWorx script, a Maestro script, etc.

The greatest benefits are derived from developing a standard model for all conversions and interfaces. By following a standard of developing modular code, much code can be reused. One of the greatest benefits accrue to the support of the interface once it is placed in Production. Because the design is known, troubleshooting and repair is greatly simplified. A simple standard for interface design is shown below:



Step One – In this step, the flat file of transactions is moved from the remote host. Most commonly, FTP is used to move the file.

Step Two – The flat file is loaded into a temporary table. This table layout contains one field for each field in the flat file and fields for each foreign key reference in the flat file.

Step Three – In step three, foreign keys are validated. For each value in the source transaction layout, The surrogate key for any foreign keys (inventory_item_id for segment1 in the inv.inventory_item_master table, for example) are . If a value exists in the source transaction for which there is no identical value in the Applications database, then a referential integrity error would occur if that transaction were to be loaded in the Open System Interface table. It is more desirable to eliminate bad records as early in the interface process as possible. For example, if the interface loads requisitions, and a transaction contains a part number which is not found in the item master, then the requisition would generate an error in the open systems interface.

Testing the interface

There are three phases of testing the interface should go through:

1. **Unit Test**

This is where the developer tests the Interface to ensure that it meets the specification. Inputs are measured against outputs. The fault tolerance of the interface is ascertained.

2. **Integration Test**

This is where the interface is measured against the other business processes. The scheduling of the interface is validated. Its ability to operate in the Oracle Applications technical environment is determined.

3. **QA Testing**

In this phase, the team practices putting the interface into production. The installation scripts are tested. As the user community participates in QA testing, the interface will run just as it would in production.

There is a temptation to shorten testing time periods. Since people learn iteratively, testing provides an iterative examination of the interface in various situations. Experience shows that the number of change requests increases, the more the interface is tested. Initially, the changes are requests to repair faulty logic. Eventually, the changes request new features that the users have only recently discovered they needed. It is best to test before the code goes into production, not after.

Moving the interface into production

The safest way to put software into production is to practice. There should be a time period when the implementers perfect installing a particular piece of software. Not only does it validate the proper functioning of the install scripts, it gives the team an idea of how long it takes to implement. It may be discovered that the implementation time window isn't large enough. It is better to discover the windows is too small BEFORE the big day arrives. The software should be put into a single package containing:

1. The source code for the interface. In addition to the actual conversion or interface code, this should include any scripts that build tables, build indexes, populate tables, and so on. If simplicity is desired, it could contain Oracle Export files (*.dmp) of basic tables required.
2. The script that installs, builds, parses, and generates the interface. This script should run the installation steps in the proper order.
3. Documentation, including changes requests.

The package should be stand-alone. It should contain everything required to install the software. This package could be a .tar file, a .z file, or a zip file. Ideally, the package would be checked into a version control software package (like PVCS or SCCS) before it is implemented in Production.

Producing Interfaces Faster

It is likely that the Development Team responsible for producing the conversions and interfaces will be on the critical path. There are a number of possible reasons for this, including:

1. The team is new and inexperienced with Oracle Technology;
2. The people producing the functional requirements is running late on delivering the specifications, however, the program manager is not providing the development team a commensurate extension on their deliverables;
3. The technology selected to implement conversions and interfaces has a) a long learning curve; b) a long debug cycle; c) a time consuming development process; d) may not be intended to produce conversions and interfaces
4. The Project Plan was developed without an appreciation as to how long it would take to produce specifications, design, develop, test, and implement a solution.

As a result, the wise Implementation Team seeks out ways of producing conversions and interfaces faster. By the time the development team receives the specifications, there may only be a few days or weeks until the code needs to be in or through testing.

There are a few key things that, if implemented, will greatly facilitate productivity:

1. Hire experienced consultants – bringing the experiences gained on other implementations is critical to a short learning cycle, and thus a shorter development cycle.
2. Reusing code wherever possible – This allows code that has already been tested to be leveraged. It isn't productive for two developers to develop the same functionality twice, but it is commonly permitted on less than productive teams.
3. Purchase as much integration as you can afford – There are a number of integration solutions available. Look for Oracle CAI partners, as this ensures that the vendor has met Oracle's strict quality requirements.
4. Use a software tool to facilitate development –Again, there are many alternatives to using technology to get a productivity gain, but the choice must be made carefully.

The characteristics of the development tool that facilitates the greatest productivity:

- The tool should facilitate improvements in Data Quality – it should permit measurement and remediation of problems

- It should be able to handle a variety of requirements as well as be scalable from ten to a hundred to hundreds of transactions .The tool should provide a simple process to manage the complexity of a large number of transactions.
- The tool should have demonstrated a development cycle time much faster than manual coding.
- The tool should facilitate rapid error identification and troubleshooting
- The tool should support and enforce the development standards – Consistency of implementation is vital
- The tool should provide documentation about the conversions and interfaces. Developers are generally loath to produce documentation manually, which means the quality is usually less than desirable, and is rarely updated as changes occur.
- The tool should offer significant productivity gains – it should behave as a Virtual Employee, doing the wide range of tasks that a person would otherwise do.

About Constellar Hub

The Constellar Hub is an EAI engine that provides a maintainable solution for data transformation, data movement and interface management in any Oracle Applications environment.

The Constellar Hub utilizes an Oracle database to provide a transient data cache for the staging of data from multiple applications as well as leveraging the security, scalability and performance of the Oracle environment.

Constellar Hub has a centralized hub architecture for the development and maintenance of all inter-application interfaces across the enterprise. Constellar Hub is highly tunable to ensure the highest possible transformation speed.

In addition to the Constellar Hub, Constellar also offers Constellar Interfaces for Oracle Applications. Constellar Interfaces for Oracle Applications consists of pre-built conversion templates and interfaces that are certified for Oracle Applications. This certification is part of Oracle's Cooperative Applications Initiative so you know that the Constellar solution has been examined and certified by Oracle for use with Oracle Applications.

Using the Constellar Hub's graphical user interface and powerful Transformation Definition Language, sophisticated rules-based transformations and set-based manipulations are possible as data is moved in and out of Oracle Applications.

Many Global 2000 customers utilize Constellar Hub to support the interfacing of a wide variety of operational systems.

Other components of the Constellar product family include Constellar WarehouseBuilder and Constellar MQSeries Connectivity.

Constellar WarehouseBuilder provides sophisticated pre-aggregation of data for data warehouses, marts and other OLAP environments including Oracle Express. Constellar WarehouseBuilder uses a unique approach, called "the Dimensional Framework", to construct multi-dimensional aggregates in a single pass of the source data.

Constellar MQSeries Connectivity provides Constellar Hub with the ability to read and write messages from IBM's MQSeries, the market-leading messaging middleware. This connectivity enables Constellar Hub to integrate batch/bulk and near real-time data movement in one product.

Constellar Hub is the leading product specifically designed for EAI. Constellar is committed to providing a tool that speeds the implementation of Oracle Applications, as well as eases the challenges of integrating legacy systems with Oracle Applications.

To that end, the Constellar Corporation is partnering with leading Oracle Applications technology providers to provide the full Oracle Applications integration solution described in this white paper.

About DataMirror Corporation

DataMirror (Nasdaq: DMCX; TSE: DMC) delivers solutions that let customers integrate their data across their enterprises. DataMirror's comprehensive family of products includes advanced real-time capture, transform and flow (CTF) technology that gives customers the instant access, integration and availability they demand today across all computers in their business.

Over 1,400 companies use DataMirror to integrate their data. Real-time data drives all business. DataMirror is headquartered in Toronto, Canada, and has offices worldwide. DataMirror has been ranked in the Deloitte and Touche Fast 500 as one of the fastest growing technology companies in North America.

DataMirror[®]

www.datamirror.com

1 800 362-5955