

## Oracle eBS 11i Infrastructure

In this article we will describe the infrastructure of Oracle eBusiness Suite (eBS). In its simplest form, eBS is a 3-tier application with a client tier, Application-tier and DB-tier.

### Database-Tier

Let's start with the DB-Tier. Surprisingly, the database tier has only very little eBS specific features.

Of course we need a database (instance) and therefore an ORACLE\_HOME. But the database can either be a single-instance or a RAC-installation and all Oracle RDBMS features are transparently available for eBS.

The management of the RDBMS Installation is also independent of eBS.

### DB-Tier filesystem

Let's start with the filesystem on the DB-Tier. Of course there is an Oracle\_home installation needed, for the RDBMS-Instance. This will be installed during installation of eBS. But also a fresh installed ORACLE\_HOME can be used, with an eBS database.

In the ORACLE\_HOME, an extra directory is added. The Appsutil directory. This directory contains the software and data needed for running Autoconfig and Rapidclone.

All other directories are at the discretion of the eBS DBA.

### Oracle Instance

When we look at the instance to run eBS, we find a number of mandatory parameters for eBS. These are found in Metalink notes 216205.1 and 396009.1 (At the time of writing. Please verify these notes for yourself).

These parameters are recommended or mandatory based on testing by Oracle Corp. They will automatically be set by the eBS installer. But you should take note of them when you use a fresh installed ORACLE\_HOME.

Then we finally come to the contents of the database.

### The eBS Database

Let's start with the schemas in the database. Oracle eBS creates a separate schema for every module. The schema is named as the short\_name of the module, for example AP (Oracle Payables / Accounts Payable), AR (Oracle Receivables / Oracle Receivables).

There is a separate schema for the Application owner APPS.

The Application schemas contain the tables, indexes and sequences for the different applications. All objects in these schemas (except indexes, of course) have a synonym in the APPS Schema. In the APPS Schema we also find all PL/SQL objects, views and Materialized Views.

A major part of eBS is written in PL/SQL. All PL/SQL objects are also installed in the APPS Schema.

User sessions within eBS will usually run in the APPS Schema as well.

That brings us to an extra schema in the database: APPLSYSPUB. This schema has access to some of the eBS tables and packages, that allow it to validate eBS logins and start an APPS-session based on that login information. We will see the details of this later on.

Before release 11.5.2 every schema had its own tablespace. However, the number of modules for eBS (and with that the number of schemas) is ever increasing. So managing the database became more and more complex. In 11.5.2 Oracle introduced the Oracle Applications Tablespace Model (OATM). Within this model, the tablespaces in eBS are based on functionality, rather than schemas.

In this model, we see the following tablespaces:

APPS\_TS\_TX\_DATA – Containing all transaction tables, Materialized Views and IOT's

APPS\_TS\_TX\_IDX – Containing all indexes for transaction tables

APPS\_TS\_SEED – Containing the tables and indexes with seeded data (as opposed to transaction data).

APPS\_TS\_INTERFACE – For Open Interface tables

APPS\_TS\_SUMMARY – Contains summary tables for several modules (AR, PA, BIM, etc)

APPS\_TS\_NOLOGGING – For tables and objects that are created with the NOLOGGING option

APPS\_TS\_ARCHIVE – Containing archive and history tables (and indexes)

APPS\_TS\_QUEUES – Containing the AQ (Advanced Queuing) objects

APPS\_TS\_MEDIA – Containing tables with LOB's. For media objects or documents.

The Undo and Temp tablespaces are not part of the tablespace model.

## Application Tier

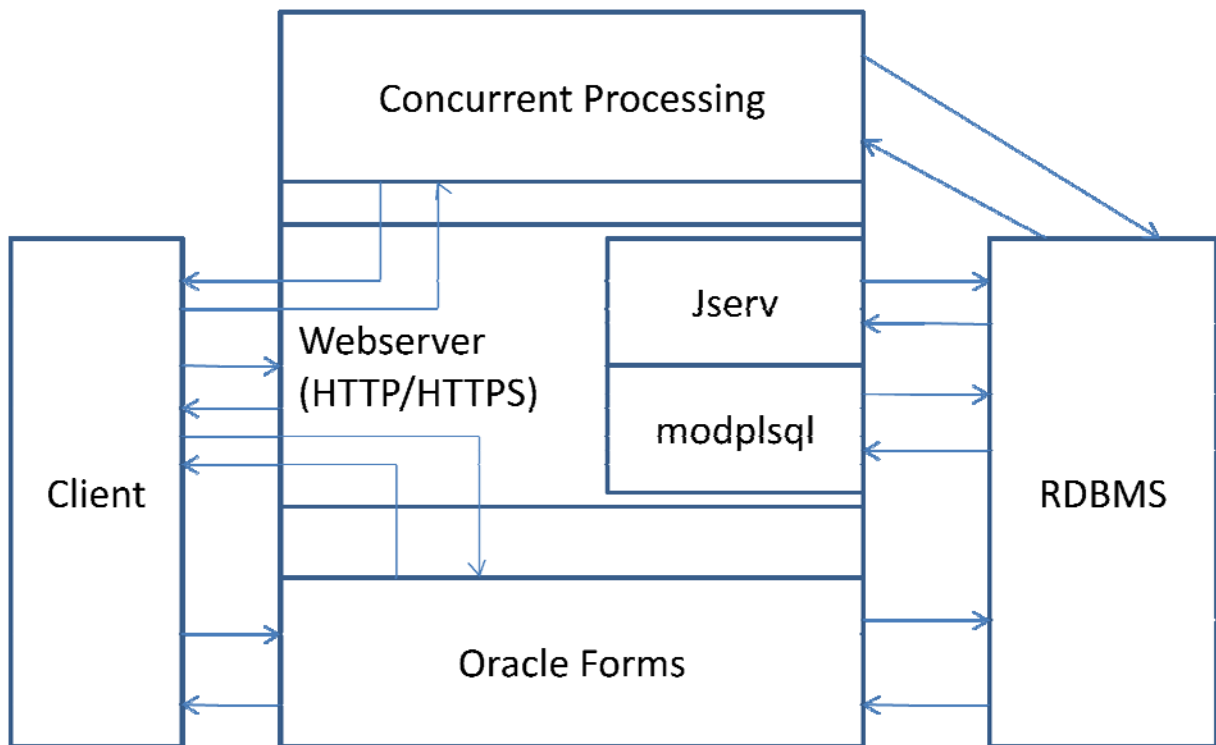
Now it's time to look at the Application Tier. In fact the Application Tier consists of 3 different services: Web service, Forms service and Concurrent Processing. In 11i installations, there is also an Administration service.

The Application Tier significantly changed from R11i to R12. We'll discuss the 11i Apps Tier shortly, and then discuss the R12 tier in more detail.

## The 11i infrastructure

Both the R11 and R12 infrastructure consist of a Web-service, a Forms Service and a Concurrent Processing part.

We will be discussing the different services here. The following picture shows all components and their communications. You might want to keep it for reference during this article.

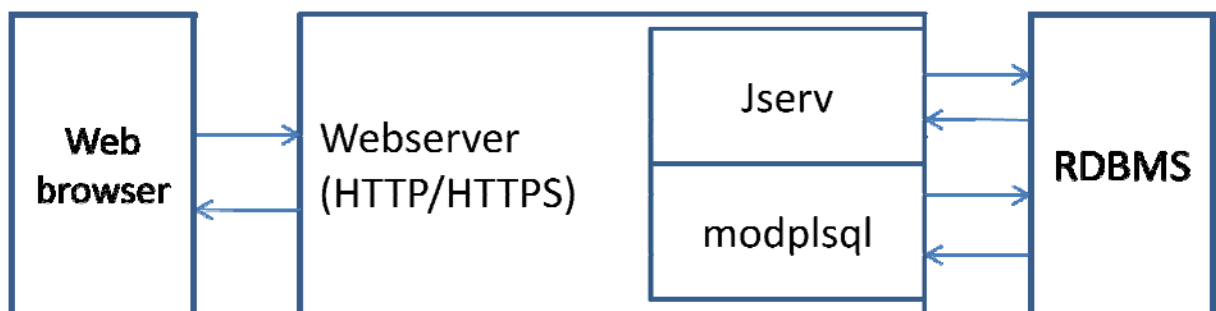


### R11i Web Service

For 11i, the web tier is built on Oracle iAS 9i. The iAS installation provides webservices (Apache HTTP/HTTPS), a Java Runtime Engine (JSERV) and a PL/SQL engine (modplsqli).

The web service also acts a gateway for the Concurrent Request log and output files. And it is the first point of access when starting a forms session. (When using a socket forms connection, when using a forms servlet the web tier will host the forms process).

A detail from the picture above shows the iAS structure.



The core of iAS is the web server. This is the front-end for the client. Requests can also be forwarded to and from the forms server and the concurrent processing. We'll see that in the next paragraphs.

Within the iAS Jserv and modplsql are plugins. They are the only components that communicate with the database. When they are called, they execute java (Jserv) or PL/SQL (modplsql) and return an html page. This page is then sent to the client through the HTTP service.

The Jserv delivers a Java Runtime Environment. In the Jserv, java servlets can be run. Also the JSP-files are executed in the Jserv. A JSP-file (Java Server Page) is a page with java code that returns an html-page (similar to the way scripting languages like PhP work). The java part is executed in Jserv, which returns the html to the webserver. The webserver redirects the html to the client.

Let's take a closer look at the components and their executions:

### Webserver

The webserver is based on the regular Apache 2 webserver. The configuration file is also equal to the Apache config file. The configuration is set in httpd.conf (or httpds.conf for SSL).

Instead of starting the webserver through \$APACHE\_HOME/bin/httpdctl, we start through \$APACHE\_HOME/apachectl. The default port number used for eBS 11i is 8000. This is part of port-pool 0. For different port-pools, the port number is increased. So for port-pool 1 the webserver runs on port 8001.

The root directory for the webserver is set to \$OA\_HTML, which is by default \$COMMON\_TOP/html. This directory contains all \*.html files for eBS.

A number of virtual directories are set up within eBS.

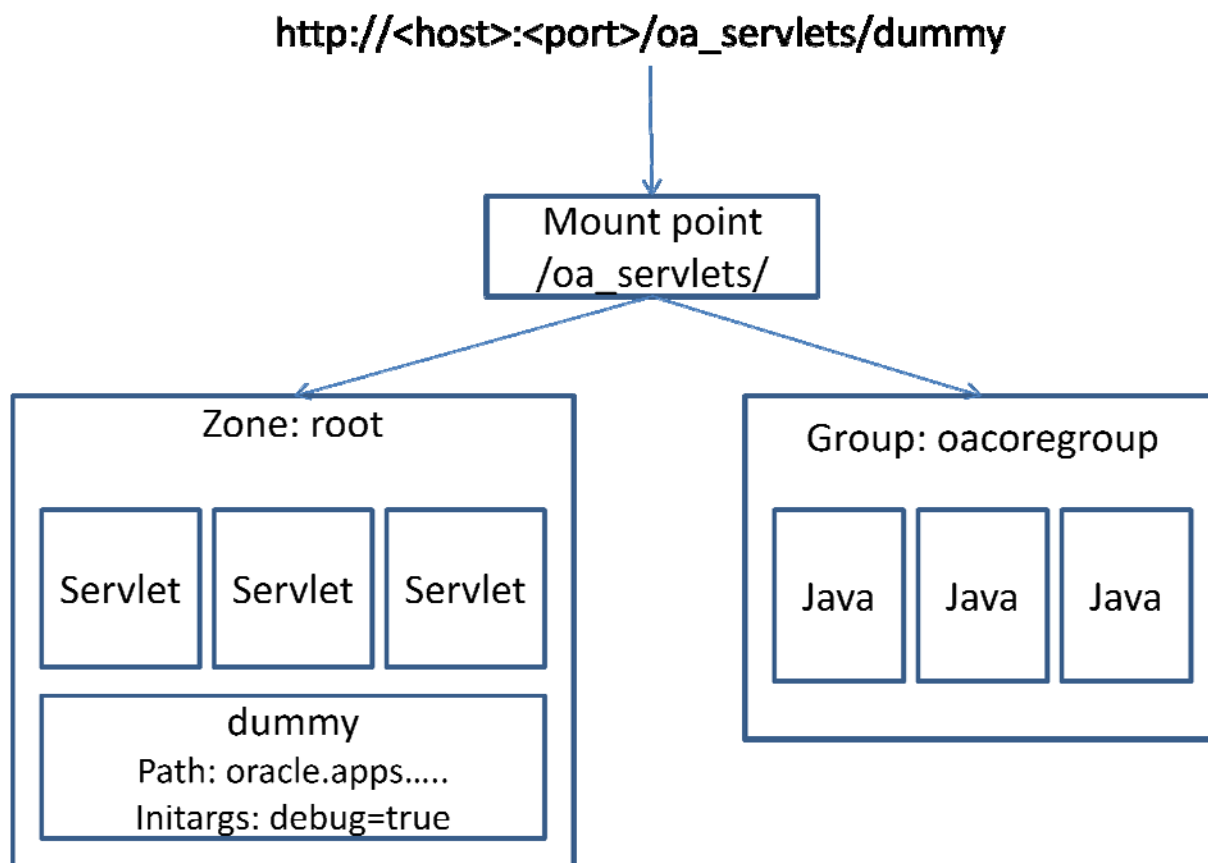
### JServ

As mentioned before, java code is executed by Jserv. Jserv is a java servlet engine. That means that it can run both servlets and jsp-files.

These servlets are mostly located in the \$JAVA\_TOP. The \*.jsp files are located in the \$OA\_HTML directory.

One of the options of Jserv is to create a database connection to the rdbms database. This is done by a JDBC Thin Client connection.

Before we look at the configuration for Jserv, examine the following picture.



Within Jserv, we can define different java environments, called zones. These zones are configured with different servlets or java archives (jar-files). Each zone is configured with its own configuration file. Within the zone the startup parameters (initargs) for the servlet are defined.

On the other side of the picture, you see a group. All java processes within Jserv are grouped together. You must define at least one group. The default group is 'OACoreGroup'. Within each group, we create one or more processes that will be mapped to our zones.

This mapping is done by mounting the zones and the groups to different logical directories. In the picture, a mountpoint is created: `/oa_servlets/`. It refers to the group 'OACoreGroup', which holds 3 java processes. And it is mapped to zone 'root', which includes the servlet 'dummy' with a set of startup parameters.

When iAS receives a call to the virtual directory `/oa_servlets/` it will be recognized as a Jserv mount point and the request will be forwarded to Jserv. In this example Jserv has 3 java processes in the group for this mount point. And they will be able to run all the servlets in the zone.

Sounds complicated? Take a look at the following configurations:

Jserv.conf:

```
ApJServGroup OACoreGroup 3 1 /etc/oracle/iAS/Jserv/etc/jserv.properties
ApJServGroupMount /oa_servlets balance://OACoreGroup/root
```

Jserv.properties:

```
wrapper.bin=/opt/oracle/iAS/Apache/Apache/bin/java.sh
```

```
zones=root<host>
root<host>.properties = /etc/oracle/ias/Jserv/etc/zone.properties
```

### Zone.properties

```
servlet.Dummy.initArgs=message=I'm a dummy servlet
```

Within jserv.conf we define a group called OACoreGroup. This group is running 3 Java processes. And the definition of the group is in the jserv.properties file. The 1 indicates the weight for load-balancing with multiple groups.

Then we mount the zone 'root' to the group 'OACoreGroup'. This mount point is linked to the virtual directory /oa\_servlets/.

The virtual directory is used for redirection to the JServ. When a request is made to the virtual directory jserv will be called. The part of the URL after the virtual directory is the path to the servlet. This path will be searched for in the \$CLASSPATH.

When the java servlets need to connect to the database, they can build a connection using JDBC. The access information is stored in a \*.dbc file in \$FND\_TOP/secure. The dbc-file is referred to in the parameters for the zone.

## Modplsql

Let's take a look at the modplsql module. This module is designed to run pl/sql procedures within the database. The connection is based on the wdbsvr.app file. This file contains the DAD (Database Access Descriptor), including the access data to the eBS database.

The module is also called through a virtual directory. For example `http://<host>:<port>/pls/TESTDB/dummy`. /pls/ is the virtual directory that refers to modplsql. TESTDB is the name of the DAD and dummy is the name of a pl/sql procedure accessible for the db-user from the DAD.

That concludes the 9i iAS module for now.

## Formserver

Let's take a look at the forms server.

Oracle forms can be set up in two ways, socket connection and servlet. The default is socket connection. With a socket connection, a separate forms server and dedicated forms processes are used. For the servlet connection, a java servlet is called within the iAS.

The formserver itself is installed in the 8.0.9 ORACLE\_HOME. Forms has a forms server, and one or more client processes. The forms server is started with f60svr. It will spawn a f60webmx process for every client session connecting.

On the server side, forms are run in the forms client processes. On the client side, they are run in a java applet. When the client clicks a forms based function in the Personal Home Page, it calls an URL that refers to the forms client executable in the 8.0.6 ORACLE\_HOME.

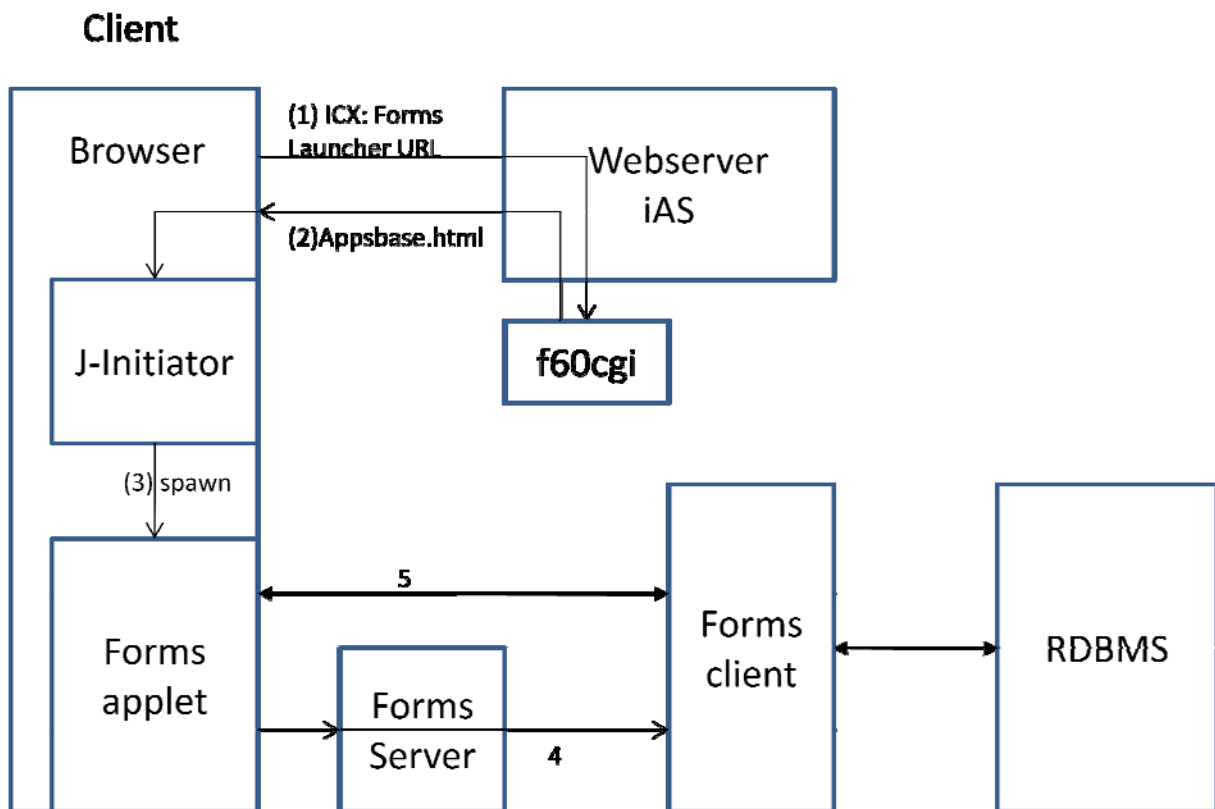
This URL is taken from the profile option 'ICX: Forms Launcher', and the default value is like 'http://<server>:<port>/dev60cgi/f60cgi'. The parameters referring to the function being clicked are added to this URL as parameters. (i.e. the name of the form to be started)

When this URL is called, the webserver will execute the executable f60cgi. This executable returns a HTML page to the client. This page is called the 'Base HTML' for this forms server. (by default this is \$OA\_HTML/US/appsbase.htm)

This HTML page calls the J-initiator plugin (or the native JVM when configured). It also includes the parameters to connect to the forms server and the name of the form to start.

The J-initiator will start an applet on the client, which connects to the forms listener process. The forms listener process then assigns a dedicated forms client process.

At this point the whole chain looks like this:



The configuration for the forms server is in the appsweb.cfg file in (\$OA\_HTML/bin). This file contains the basic coloring scheme for the forms server, the forms settings and the referral information to the J-Initiator plugin. The plugin on the client side is called through its class-id, which is also set in the appsweb.cfg.

## Concurrent Managers

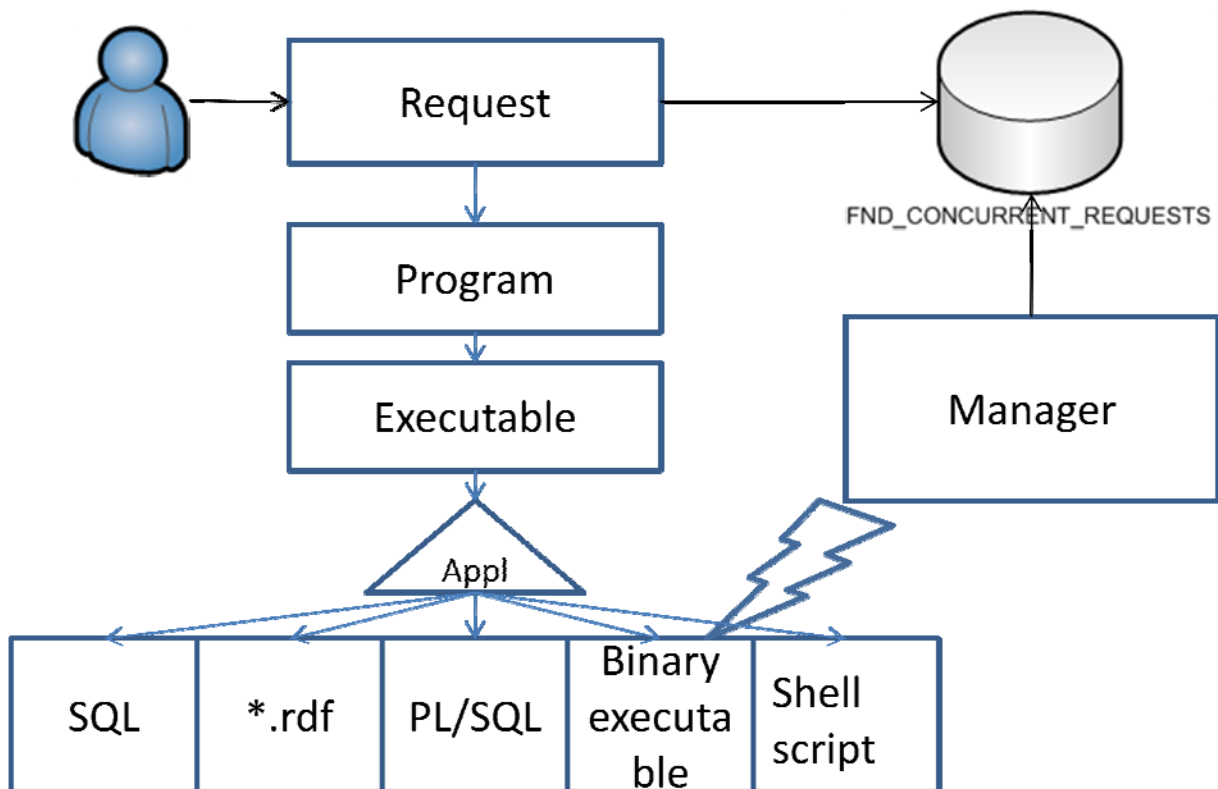
The last part of the application tier is the concurrent processing part. The concurrent managers are used to execute background and batch processes.

Different executables including host-command files, pl/sql procedures, Oracle Reports, SQL\*Loader control files and Binary executables can be defined to be run as concurrent programs. Parameters are also optionally defined with the concurrent programs.

The executable files are defined separate from the concurrent programs. So an executable can be run as different programs with different parameters.

The programs are executed through 'Requests'. A request is started as a concurrent program and the values of its parameters. It can be scheduled to start at a specific time, or in a specific schedule. The output of the program can be sent to a printer. It is also available through the application.

This picture shows the relation between programs, requests and managers.



## The managers

We'll take a closer look at the concurrent programs later. Let's first look at the concurrent managers. There are concurrent managers and transaction managers. Also a number of control managers are defined.

We'll start with the 'Internal Manager'. This is the first manager to be started. Its purpose is to control the stopping and (re-)starting of the other managers. When Generic Service Management is

enabled (default as of 11.5.7), it delegates to the 'Service Managers'. On every node where concurrent processing is enabled, a 'Service Manager' is started. However, only one Internal Manager is running at any time.

The other concurrent managers are defined with a work shift that controls how many processes a concurrent manager should have at certain times. The work shifts consist of a time-range and a number of processes. The Service Managers (or Internal Manager) will start and stop processes according to these work shifts.

Another part of the setup of concurrent managers is their specialization rules. The specialization rules indicate which programs are valid for a concurrent manager, or are excluded for that manager. They work on an include/exclude principle. When programs are included for that manager, the manager can only run those programs. When programs are excluded, the manager can run any program except the excluded ones.

When a request (to run a program) is submitted from eBS, it will be placed in FND\_CONCURRENT\_REQUESTS with a status\_code 'I' (The eBS forms have fewer statuses than the codes in the table). The manager processes will query this table for requests that they are eligible to run.

Once a manager process finds a request with status\_code 'I', which it is eligible to run then it will put the request on its own queue. It will then run the executable with the defined parameters. The logfile and outputfile are written to the filesystem in \$APPLCSF/\$APPLLOG resp. \$APPLCSF/\$APPLOUT.

There are some special cases that need to be discussed. The first is the incompatibility. Concurrent Programs can be made incompatible with each other. That means that they cannot run at the same time. Once a program is started that is defined as incompatible with another, it will be automatically put on the queue for the 'Conflict Resolution Manager'. This special manager will check if any incompatible program is running or 'Pending' with code 'I'. If so, it will hold the request on its own queue. If no incompatible program is running or 'Pending', then it will set the status of the request to 'Pending'.

Another special case are the 'Transaction Managers'. They are started and stopped the same way as the other concurrent managers. But they do not use the request queue. Transaction managers are called online from the eBS forms. And they execute a limited number of programs. These programs defined within their executables. They are called through the 'FND\_TRANSACTION.SYNCHRONOUS' procedure, which uses the 'DBMS\_PIPE' package.

## The programs

It's time to look at the concurrent programs. As mentioned before, a concurrent program is an instantiation of an executable. The executable is defined with a short name, an application (module), a filename and a method.

The short name will uniquely identify the executable. The other data is needed to determine what should be run for this executable. If the executable is an OS-based program, the application will be used to derive the directory on the file system where the executable is found.

When the executable is defined as PL/SQL, the filename will contain the procedure that needs to be run.

The concurrent program is defined as the executable with an (optional) set of parameters. It also has some properties for the printing of the output (print-style, pre-defined printer, size of the output).

Depending on the type of executable that needs to be run, the parameters will be sent to the executable ordered or named. For PL/SQL and host files, the parameters are ordered. And the order in which they are defined in the form defines how they will be sent to the executable. For reports the parameters are named, which means they are sent as <parameter>=<value>, ....

After the request has finished, the request table 'FND\_CONCURRENT\_REQUESTS' will be updated with the status information and a reference to the log- and output file. During the execution of the request, a status\_code and phase\_code are updated. The exact values of these fields are described in one of the next articles. That will go deeper into concurrent processing.

## The output

The log- and output file from the requests are written to the directories \$APPLCSF/\$APPLLOG and \$APPLCSF/\$APPLOUT. But they of course also need to be made available to the user. This is done through the 'Applications Report Review Agent'.

There is quite a lot of setup that can be done for the whole process. But within the scope of this article, we'll only look at the basic architecture.

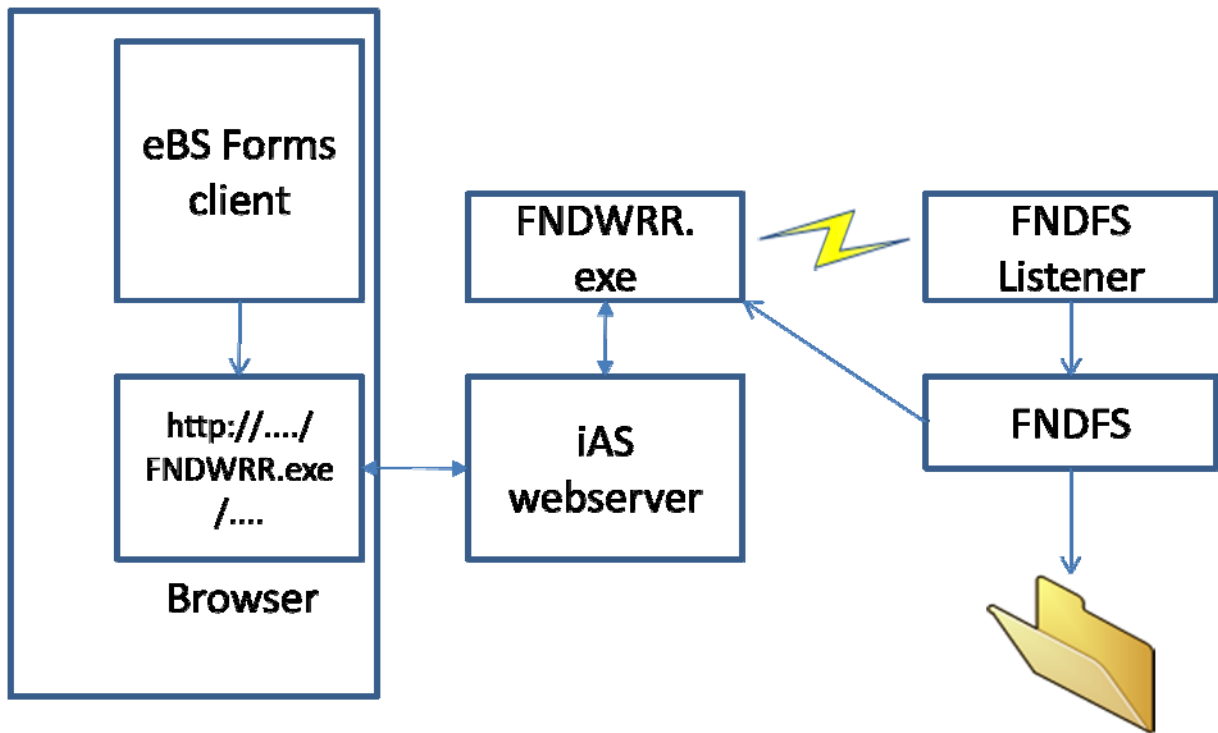
From the eBS form the log and output file are available from 2 buttons. These buttons call the web server for 'FNDWRR.exe' (.exe on both Windows and Unix).

FNDWRR.exe is a cgi-executable that will call the 'FNDFS listener'.

This is an 8.0.6 TNS-listener in the eBS ORACLE\_HOME on the eBS application tier. One of the less known features of the TNS-listeners is that they can do more than create database connections.

In the listener.ora, you can define a program to be called when a connection is made on a certain tns-entry. That feature is used for the FNDFS listener. When it is called, it will redirect traffic to the 'FNDFS' (FND File System) executable. This executable will read the requested file from the file-system and send it to FNDWRR.exe.

Again, we have a schema to show the whole flow:



This complex retrieval is of course needed because the concurrent processing tier can be separate from the forms and web-tiers.