

Introduction to Autoconfig/Customizing Autoconfig

Name: amit poddar

Company: *yale university*

Introduction:

Oracle Applications 11i consists of lots of configuration files and lots of profile options, which need to be all correctly set for the applications to work. Managing these profile options and configuration files manually is error prone. Autoconfig is a tool provided by the Oracle E-Business Suite to maintain these files and profile options. In this article, you will see how autoconfig works, what it does, what it does not do, over and above how to customize it to meet our requirements.

Background:

Oracle Applications has a complex architecture with many pieces of technology plugged in together. For example Apache Web server, Apache Jserv, Forms Listener servlet (or forms server) etc. Each of these pieces of technology has its own set of configuration files (and there are lots of them), which need to be set up correctly for them to function. Moreover Oracle Applications uses many profile options (e.g. Applications Web Agent, Applications Framework Agent etc) and other bits of information in the database to make the navigation seamless. These also need to be set correctly for the whole architecture to function seamlessly.

Before autoconfig these configuration files and profile options had to be maintained manually. If there were a need to change some configuration option, such as web server port or web domain name, the DBA would have to change all the files and profile options manually. This was quite error prone on two fronts. One the DBA needs to know and remember all the places where the change has to be made and secondly it introduces significant risks of typographical or other errors. These errors will (note I say will not may) result in long but unnecessary downtime.

This manual process is no longer recommended or supported by Oracle.

What is AutoConfig?

In short autoconfig is an 11i configuration tool (set of java classes run from perl/shell scripts) introduced in 11.5.4, which obviates the need to manually maintain these configuration files and profile options.

Autoconfig maintains the environment specific values for the 11i environment in an xml file (stored in \$APPL_TOP/admin). When autoconfig is run it overwrites the existing configuration files with new ones that it creates by merging the templates and the context file, it also updates the database by running the script it instantiates (merging a template with the values in the context file is termed as instantiating the template in Oracle parlance) from the script templates. That's essentially what autoconfig does i.e. Instantiating configuration files from templates and updating database with values from the xml file. It does not move directories, check for availability of ports, update DNS entries and so on. These have to be done by you before running autoconfig.

How does it work?

Each configuration file has one (sometimes two, one for NT and one for UNIX) corresponding template file (provided by autoconfig patches, stored in \$PROD_TOP/admin/template directory). Profile options and other instance specific information in the database is maintained by many sql scripts, called from wrapper shell/perl scripts. These scripts also have corresponding template files (also provided by autoconfig patches, stored in \$PROD_TOP/admin/templates). In these template files all the environment specific values are replaced by placeholders (like "%s_webhost%").

For e.g.
Following entry in httpd.conf
Timeout 300

is replaced by following in its template \$FND_TOP/admin/template/httpd_ux_ias1022.conf
Timeout %s_ohstimeout%

Environment specific values for this placeholder is stored in an environment specific xml file (called application context file) stored in \$APPL_TOP/admin.

For e.g.
the above placeholder the value stored in the xml file is:
<ohstimeout oa_var="s_ohstimeout">300</ohstimeout>

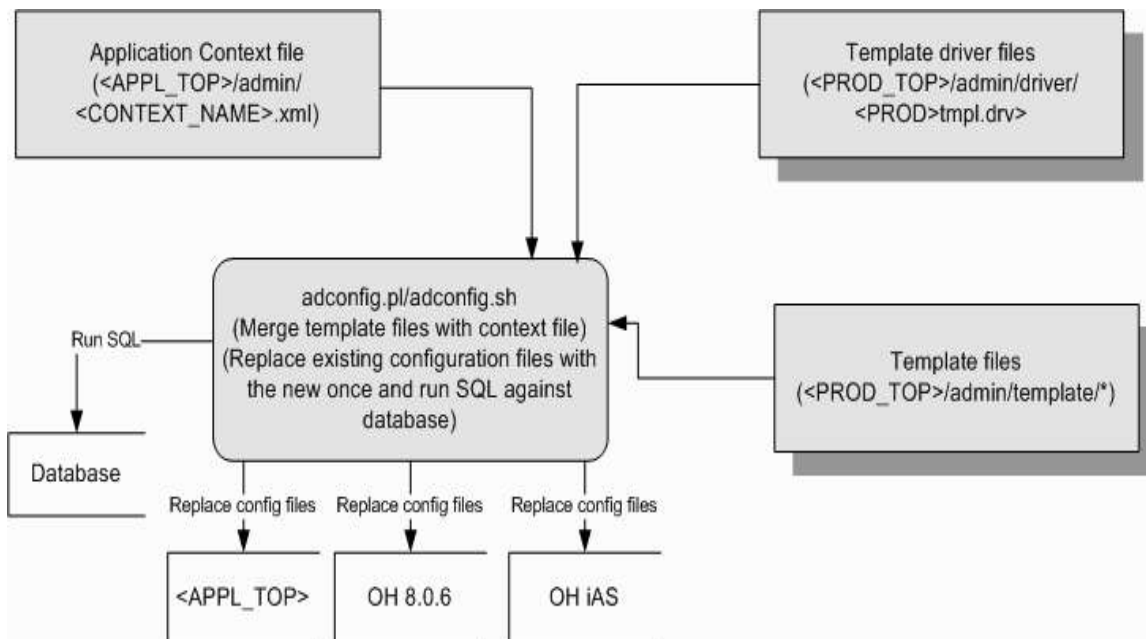
For

Each placeholder has a corresponding xml node in the context file. The placeholder name is the value for the "oa_var" attribute of the xml node, the placeholder name without the prefix "s_" (generally) is the node name for the xml node and the value is stored as a child text node. These xml nodes are termed as context variables by Oracle documentation and each node primarily is identified by its oa_var attribute value. So effectively we can say that in the above case we replace "300" by context variable "ohstimeout" whose oa_var value is "s_ohstimeout".

Configuration files can be easily created from the template file by just replacing all the placeholders with the corresponding values from the context file. This process is termed as "instantiating the template" in Oracle documentation.

Driver files (stored in \$PROD_TOP/admin/driver) store information about what to do with each template (e.g. instantiate it and replace the existing configuration file with it, instantiate it and run it). These files are named as <PROD_NAME>tpl.drv (e.g. adtpl.drv, fndtpl.drv etc.) They contain one line for each template they manage.

When autoconfig (adconfig.pl or adconfig.sh) runs it just processes the driver file for each product, line by line doing what the driver file instructs it to do. The order of execution of each line is not sequential as we shall see later. The diagram below depicts the autoconfig functionality.



The Context File:

Oracle stores all the environment specific values in an xml file stored in \$APPL_TOP/admin directory, which they call an application context file. This file is created by running adbdxml.sh/adbdxml.pl (Located in \$AD_TOP/bin). Adbdxml.sh in turn runs oracle.apps.ad.context.GenerateContext java class.

Context file is generated by plugging in environment specific values in the context file template (don't confuse this template with the configuration file templates) \$AD_TOP/admin/template/adxmcltx.tmp. In older versions of 11i these replacement values were gotten from config.txt file (created by rapidwiz during installation), but in later versions, these values are gotten from the information, in existing configuration files and the database. Adbdxml.sh/adbdxml.pl (in newer versions of autoconfig) creates a detailed log file, detailing the source for each context variable.

Following is an example entry for the context variable "domainname" from the log file of adbdxml.pl run:

```
[ s_domainname ]
SEVERITY           : INFO
SOURCE SEARCHED   : Using System commands to get Domain name value
SEARCH RESULTS    : Domainname could not be found using system commands.
SOURCE SEARCHED   : Database table - GLOBAL_NAME
SEARCH RESULTS    : Query on database table GLOBAL_NAME returned - XXXX.WORLD;
                  : Table is hosting incorrect value for domain.
SOURCE SEARCHED   : Prompting the user for the domain name.
SEARCH RESULTS    : Value accepted from the user - its.yale.edu
VALUE ASSIGNED    : its.yale.edu
USER ACTION       : No action required
```

The above entry shows the sources searched, search results for each search and the final assignment for the context variable domainname. As you can see adbdxml.pl provides the search information in great detail in its log file. This log file helps a lot in debugging adbdxml.pl behavior.

Each placeholder in the template file has a corresponding xml node (a context variable) in the context file. They all have a mandatory attribute called oa_var, the attribute value for this attribute is the placeholder name. Each node is primarily identified by the oa_var attribute value and can optionally have other attributes like oa_type, oa_enabled scope and default. The node's only child is a text node, which stores the placeholder's environment specific value. From here on I will refer to this text value as the "text value of the context variable". The description of these context variables along with their oa_var value can be seen in adctxinf.tmp (stored in \$AD_TOP/admin/template).

Context variables in the context file are organized in a hierarchical order, hence the choice of xml to store them. These context variables mainly lie under five top level parent nodes i.e. oa_system, oa_host, oa_install, oa_environments and oa_processes. Each top level parent node is further divided into further levels before you reach the real context variables.

To change any configuration option in the Oracle Applications environment, we need to change the value of corresponding context variable in the context file and then run autoconfig for our changes to take effect. For example if we need to change the web server port then we would change the text value of the context variable "webport" (oa_var value s_webport), and then run autoconfig. Since the context file is just an xml file, it can be updated in many ways. I will describe some of them below.

Editcontext:

OAUG Forum at COLLABORATE 06

Editcontext is a java application provided by oracle for updating the values in the context file. It is provided as a patch (patch: 2873456). Editcontext provides a list of all the updatable context variable values along with their titles. The title is not the variable name or value of oa_var attribute, but it is derived from adctxinf.xml which comes with the editcontext patch and stores the title along with the oa_var value for each context variable. Editcontext only lists those context variables, which are listed in its repository (adxmlctx.tmp). If you find that, not all variables are listed, then the version of this file is probably lower than the adxmlctx.tmp stored in \$AD_TOP/admin/template. In that case just copy this file from \$AD_TOP/admn/template to the editcontext repository (i.e. <patch unzip directory>/editcontext/etc).

After launching editcontext, we have to find the context variable that we want to update. Once we find it, we can just change the value and save the new value to the context file (To be precise editcontext rewrites the whole context file instead of just updating one value). Since the list of context variables is quite long, it may take a while to find the desired context variable in the list.

Oracle Applications Manager:

Oracle Applications Manager is a web-based portal for managing Oracle Applications. Along with many other features, this product has a section for updating the context files. This product lists everything for each context variable i.e. name, oa_var attribute value, description, and real updateable value. Moreover instead of giving us a long list of context variables to search from, it divides the context file into many sections and each section is displayed in the xml tree format making it easier to find a particular context variable. It also provides us with the option of adding custom context variables (which we will look at in the customization section).

One thing to remember while updating context variables using OAM is that what it displays is gotten; by parsing the context file stored in fnd_oam_context_files table (Autoconfig at each run uploads the context file in this table). When we try to update the context file, OAM first update's the status flag from 'S' to 'H'(History) for our context file record, it then inserts another row for the same context file with status 'S'. (OAM inserts another row instead of updating the existing row to maintain change history for each context file) It then requests the specific node's FNDFS listener for updating the file on the file system (autoconfig uses the file on the file system). So we have to make sure that the FNDFS listeners are running on all the nodes before using OAM to update the context file.

With all the features of Oracle Applications Manager mentioned above, updating context files using Oracle Applications Manager seems to be the direction Oracle is trying to move.

EditContext and Oracle Application Manager are good for updating context file if you need to update one or two context variables. If you need to modify quite a few variables (for e.g. if you need to update all the ports in the context file) then using these GUI tools do become cumbersome. It would be nice to have a scripted way of updating lots of values in the context file using a data source of changes. I will describe two ways of achieving this using Oracle supplied code.

Java class oracle.apps.ad.context.UpdateContext:

This java class allows us to change one context variable at a time from the command line. But we can easily use a shell/perl/python/tcl/etc script to call this class multiple times for different context variables based on a changes file, which stores all our required changes.

For example let's say we need to change the following context variables. The first column is the oa_var values of the context variable and the second column is desired value. These values are stored in a file changes.txt.

```
s_webport 7777  
s_webport_pls 9999  
s_oprocmgr_port 9997  
s_active_webport 9996
```

OAUG Forum at COLLABORATE 06

s_forms_servlet_portrange 1801-1810
s_oacore_servlet_portrange 2801-2810

The following perl script loops through each record in changes.txt and updates the context file by calling the java class once for each line. The syntax it uses to call this java class is:

```
java oracle.apps.ad.context.UpdateContext CONTEXT NAME VALUE
```

where CONTEXT – Context File Name
NAME – context variable name
VALUE – Desired Changed value

```
#!/usr/bin/perl
```

```
use FileHandle;  
use strict;
```

```
# Changes file supplied as the first argument  
my $changes_file = $ARGV[0];
```

```
# Context file supplied as the second argument  
my $context_file = $ARGV[1];
```

```
my $fh = new FileHandle;
```

```
# Lets try to open our changes file  
# If we fail to open the changes file  
# this script would die with the  
# error message  
unless ($fh->open("< $changes_file"))  
{  
    die "Error (!) Encounterd while opening changes file:" . $changes_file;  
}
```

```
# This is the common part of our java command, which we will run  
# for each line in our changes file  
my $java_cmd = "/usr/java14/bin/java oracle.apps.ad.context.UpdateContext " . $context_file;
```

```
# This will hold the actual java command, which we will run  
my $local_java_cmd;
```

```
# Now let loop through each line in our changes file.  
# Change the java_cmd for each option  
# And run the java command
```

```
while (<$fh>)  
{  
    # Remove the new line character from each line  
    chomp($_);
```

```
    # Let split the line into the oa_var value and the  
    # changed value  
    my @changes = split //,$_;
```

```
    $local_java_cmd = $java_cmd. " " . $changes[0] . " " . $changes[1];
```

OAUG Forum at COLLABORATE 06

```

print "Running $local_java_cmd\n";
my $rc = system($local_java_cmd);

print "Successfully Updated\n" if ( $rc == 0 );
print "Failed rc = " . $rc . "\n" if ( $rc != 0 );
}

```

Using Perl Module TXK::XML:

The above technique works well but it is quite heavy on resource consumption, since we start a new jvm for every update. For every update the java code has to parse the xml file, create the DOM tree, do the update in memory and then write it back to the file system. It would be more efficient if we could just parse the xml file only once and do all our updates before writing the xml file to the file system. This can be achieved by using oracle supplied perl module TXK::XML (Found in \$AU_TOP/perl/TXK). For using TXK::XML we load the context file as as TXK::XML object, change the configuration option values by using setOAVar method of the TXK::XML object and then write the file back to the file system. The only thing to take note of is setOAVar expects a hash reference as the input variable. This hash will contain the list of context variables to be changed in a key value pair; the key is the oa_var value of the context variable and the value is the corresponding desired values.

The following perl script demonstrates this using the same example as above.

```

#!/usr/bin/perl

use FileHandle;
use strict;

use TXK::XML;
use TXK::Common;

# Changes file supplied as the first argument
my $changes_file = $ARGV[0];

# Context file supplied as the second argument
my $context_file = $ARGV[1];

my $fh = new FileHandle;

# Lets try to open our changes file
# If we fail to open the changes file
# this script would die with the
# error message
unless ($fh->open("< $changes_file"))
{
    die "Error (!) encountered while opening changes file:" . $changes_file;
}

# Create a new TXK::XML Object to load our context file
my $xmldoc = new TXK::XML;

# Load our context file. loadDocument expects a
# hash reference as a input variable instead of context file
# name as a scalar
$xml->loadDocument ( {"file" => $context_file} );

# Hash to be supplied to setOAVar method
my %changes;

```

OAUG Forum at COLLABORATE 06

```
# Now let loop through each line in our changes file.
# We will populate our changes hash as we loop through
```

```
while (<$fh>)
{
  # Remove the new line character from each line
  chomp($_);

  # Let split the line into the configuration option and the
  # changed value
  my @changes = split //,$_;

  $changes{$changes[0]} = $changes[1];
}
```

```
# Update the xml DOM tree in memory
$xmlldoc->setOAVar( \%changes );
```

```
# Write back to the context file.
# We could create another xml file
# But in this case I am just overwriting the
# existing context file
$xmlldoc->storeDocument ( {"file" => $context_file} );
```

If we use any programmatic method to update the context file, we are not allowing OAM to maintain a change history for all our context files, which is very important since context file is the central repository for all the configuration options. This can be easily taken care by updating the existing record for our context file to status 'H' in `fnfnd_oam_context_files` table and then inserting the changed context file into the table using executable `FNDCPUFC` (in `$FND_TOP/bin`).

Since the context file is just an xml file, you can basically use any xml parser or write your own xml parser to do the updates, but in my experience the above mentioned methods has been the least error prone and least time consuming. Moreover in all of the above methods I am using Oracle's code to update the xml file.

The Template Files:

Template files are files, which are merged with the context file (instantiated) to produce configuration files or scripts to update the database. They are stored in `<PROD_TOP>/admin/templates` directory. Autoconfig patches/Technology stack patches install new templates or upgrade existing templates. Now let's look at how these template files are created.

Let's take a small configuration file `REP60_<SID>.ora` (Reports server configuration file):

```
mailprofile="$Header: REP60_server.ora 115.5 2004/03/24 01:10:51 njoseph ship $"
nlssupport=yes
maxconnect=20
cachedir="/u02/app/oracle/product/8.0.6/reports60/server/cache"
cachesize=50
minengine=5
maxengine=10
initengine=5
maxidle=30
security=1
englife=50
```

OAUG Forum at COLLABORATE 06

If we replace all the environment specific values with context variables from the context file, then this is what we get:

```
mailprofile="$Header: REP60_server.ora 115.5 2004/03/24 01:10:51 njoseph ship $"
nlssupport=yes
maxconnect=20
cachedir="%s_tools_oh%/reports60/server/cache"
cachesize=50
minengine=%s_minengine%
maxengine=%s_maxengine%
initengine=0
maxidle=30
security=1
englife=50
```

And now we have a template file corresponding to the Reports server configuration file. The placeholder values enclosed by "%" correspond to the context variable in the context file. If we instantiate this template using the values from the context file we can easily recreate our configuration file. All the configuration files and sql scripts are converted to templates in similar fashion and supplied to us by autoconfig patches.

The Driver Files:

Driver files, as the name suggests drives what autoconfig does (stored in <PROD_TOP>/admin/driver). They are named as <PROD_NAME>tpl.drv i.e. adtmpl.drv, fndtmpl.drv etc. Autoconfig patches install and update these driver files. Each driver file has one line for each template it manages. Each line tells autoconfig what to do with that template. This is achieved by using a particular syntax that conveys the relevant information to autoconfig. Following is the syntax for the lines in the driver files.

<PROD> <Location> < Name> <Action> <Dest directory> <Dest file name> <File permission>

For e.g.

```
ad admin/template adconfig.txt INSTE8 <s_at>/admin adconfig.txt 600
```

Column	Description
<PROD>	Product Name
<Location>	Directory underneath <PROD_TOP> where the template is located. Generally "admin/template".
<Name>	Name of the template File
<Action>	Type of action to be performed on this template (Refer to the following table for description of different kinds of action)
<Dest directory>	Destination directory of the instantiated template file. We can use context variables enclosed in "<>" to specify the destination directory. For example if we want to specify \$APPL_TOP/admin as the destination directory we would specify it as "<s_at>/admin" in the driver file.
<Dest file name>	Destination configuration file name for the instantiated template file. We can use context variables enclosed in "<>" to specify the destination configuration file name. For example if the destination file has Database SID we can specify it as "<s_dbSid>.env" in the driver file.
<File permission>	Autoconfig generates the configuration file with the provided unix style permission.

Actions:

Action	Description
--------	-------------

INSTE8	<ul style="list-style-type: none"> • Instantiate (replace context variables with values from xml context file) • Copy the result to the <Dest directory>/<Dest file name> overwriting the existing configuration file, if one exists. • Change the permission to <File permission>
INSTE8_SETUP	<ul style="list-style-type: none"> • Instantiate (replace context variables with values from xml context file) • Copy the result to the <Dest directory>/<Dest file name> overwriting the existing file, if one exists. • Change the permission to <File permission> • Execute the instantiated file.
INSTE8_APPLY	<ul style="list-style-type: none"> • Instantiate (replace context variables with values from xml context file) • Copy the result to the <Dest directory>/<Dest file name> overwriting the existing file, if one exists. • Change the permission to <File permission> • Execute the instantiated file.
INSTE8_PRF	<ul style="list-style-type: none"> • Instantiate (replace context variables with values from xml context file) • Copy the result to the <Dest directory>/<Dest file name> overwriting the existing file, if one exists. • Change the permission to <File permission> • Execute the instantiated file.
INSTALL	<ul style="list-style-type: none"> • Instantiate the template file only if the resulting configuration file does not already exist.

Though the description of three of the actions is same, they are there to get different execution order within autoconfig's execution of each driver file. The execution order in which autoconfig processes each line is INSTE8, INSTALL, INSTE8_SETUP, INSTE8_APPLY and INSTE8_PRF. These actions mark the different phases of autoconfig i.e. Setup phase, Apply phase and Profile phase. It's also possible to run autoconfig only up to a particular phase.

The driver file syntax also supports if then else branching based on type of node (i.e. web, admin, forms etc), which is useful in a multi node installation. The syntax also supports the branching based on the platform type (Unix or Windows). This allows having multiple templates for each configuration file and instantiating a particular template based on the platform.

```
For e.g.
if platform NT
  ad admin/template APPLSYS_nt.env INSTE8 <s_at> <s_dbSid>.env
endif
```

or

```
if installation-type admin node nodedev forms formsdev web webdev
  ad admin/template adconfig.txt INSTE8 <s_at>/admin adconfig.txt 600
end if
```

The driver file syntax also supports including other driver files.

```
For e.g
#include fnd admin/driver fndtmpl.driv
```

When autoconfig (adconfig.pl or adconfig.sh) runs it just processes each product's driver file respecting the, if then else branching as it processes the lines in the driver files. It also processes all included driver files recursively.

Customizing AutoConfig:

Autoconfig is a very effective configuration management tool, but there are times when it falls short of meeting all our requirements and it becomes necessary to customize it. Following are some of the examples where you would need to customize autoconfig.

You need to add another zone to the Jserv. In this case you would have to customize oracle supplied autoconfig template which corresponds to jserv.properties configuration file, to add entries for another zone, you would also have to create a custom template for the properties file for this new custom zone. You may also need to add some custom context variables to the context file for this new custom template.

You have some custom product tops. You would have to add these custom product top environment variables to the formservlet.ini file to access the custom forms. In this case you will have to customize the formservlet.ini's autoconfig template to add your custom product top variables. You can construct these product tops based on the \$APPL_TOP's context variable i.e. %s_at%/<custom_product_top> or you can create new context variable for each custom product top. In that case you will have to add these to the context file as custom context variables.

You have developed a custom application and you want autoconfig to maintain all the configuration files of this custom application. In this case you will have to create custom driver files, custom templates and probably some custom context variables.

All the above examples show that there are three types of customizations that should satisfy most of the custom requirements:

- Adding custom context variables to the context file
- Customizing an AutoConfig template file delivered by Oracle.
- Creating custom Autoconfig templates and custom Autoconfig driver.

Adding Custom Context Variables:

Generally most of the environment specific values have already been converted to context variables, but there are still some of them which are still hard coded in the template files till ADX.F.

For e.g. consider

The reports server configurations file template i.e. \$AD_TOP/admin/template/REP60_server.ora

```
mailprofile="$Header: REP60_server.ora 115.6 2004/11/30 05:20:55 ptiwari ship $"
nlssupport=yes
maxconnect=20
cachedir="%s_806config_home%/reports60/server/cache"
cachesize=50
minengine=%s_minengine%
maxengine=%s_maxengine%
initengine=0
maxidle=30
security=1
englife=50
```

In the above case some of the parameters like maxconnect, cachesize, initengine, maxidle, initengine and englife are hard coded in the template file. If we want to convert them also to context variables then we will have to add some custom context variables to the context file.

OAUG Forum at COLLABORATE 06

Adding a context variable to the context file means, adding a node in the context file with mandatory attribute `oa_var` and optional attributes `oa_type`, `oa_enabled` and `scope`. This node will have only one child node that will be a text node holding the value of the context variable. Autoconfig doesn't care where or at what level this node is put in the file. So to make this work you can use any xml parser and add the node to the context file at any position you like and it will serve the purpose.

But there are two drawbacks to this method. One this would cause maintenance problem since it will be difficult to differentiate between custom and seeded context variables and secondly oracle's software is not aware of this customization, which means whenever we run `adbldxml.pl` or `adclonctx.pl` to clone the context file, we will lose our custom entries.

The first problem can be easily overcome by adding all custom context variables under a top level xml node `oa_customized` and starting all the custom node `oa_var` values from "c_". Now we can easily distinguish between custom and seeded context variables and it's easy to maintain since we know where all the customizations lie in the context file. In the context file section, I mentioned that the context file is created by plugging in instance specific values into the context file template `$AD_TOP/admin/adxmlctx.tmp`. So adding xml node `oa_customized` and all our custom variables under `oa_customized` to this template would take care of the second problem too.

Now let's look at different ways of adding custom context variables.

Oracle Application Manager:

Oracle Application Manager from version 2.1 onwards provides us with extensive ways to manipulate the context files. Updating the context variables is one of them, which we looked at earlier. It also provides us with the option of adding custom context variables. Adding custom context variables in OAM results in following sequence of events:

- OAM saves the custom context variable details like name, attribute `oa_var` value, attribute `oa_type` value, title and description into table `fnf_oam_context_custom`.
- OAM updates the existing row's status to 'H' (History) for `adxmlctx.tmp` in `fnf_oam_context_files`.
- OAM inserts a new row for `adxmlctx.tmp` with status 'S'. This new `adxmlctx.tmp` has our custom context variable added under node `oa_customized`.
- OAM Requests FNDFS listener of the destination node to update the `adxmlctx.tmp` on the file system.
- OAM Repeats the process for all the context files with `status='S'` and `context_type='A'`

Using perl module TXK::XML:

Oracle Application Manager is indeed a neat way to add custom context variables but there are a few problems. We have to add one variable at a time, so in our report server configuration file example, for replacing `maxconnect`, `cachesize`, `maxidle`, `initengine` and `englife` we will have to repeat the process 5 times. This can be quite cumbersome as the number of variables to be added increase. Secondly we need access to Oracle Applications but there are times (like configuring a clone) when we want to add custom context variables but OAM is unavailable. This calls for a programmatic method to add custom context variables.

Using perl module `TXK::XML` we can write a small perl program to add many context variables at one shot without needing any access to Oracle Application Manager. For doing this we have to first load the context file and the context template file as `TXK::XML` object and then use `addNode` method of `TXK::XML` package(class for OOP enthusiasts) to add the custom nodes. The only thing to note is that `addNode` expects a reference to a hash containing all the information about the node to be added. I am going to show you a piece of code for adding custom variables. It will add the custom variables mentioned in the above example of reports server configuration file. Following table details the information about the context variables we are adding.

OAUG Forum at COLLABORATE 06

Variable Name	Attribute pair	Value	Parent Node
oa_customized			oa_context
Maxconnect	oa_var=c_maxconnect, scope=custom	20	oa_customized
Cachesize	oa_var=c_cachesize scope=custom	50	oa_customized
Maxidle	oa_var=c_maxidle, scope=custom	30	oa_customized
Englife	oa_var=c_englife, scope=custom	50	oa_customized
initengine	oa_var=c_initengine scope=custom	5	oa_customized

And finally following is the piece of code which adds the above context variables to the context file supplied as the first argument. It will also add the same to the context file template. The comments in the code explain the details for each line of code.

```

use strict;

use TXK::XML;
use TXK::Common;
use TXK::Runtime;

use Data::Dumper;

# Context file is passed as first argument

my $context_file = $ARGV[0];

# Pass $AD_TOP as second argument. We expect
# AD_TOP to be passed as argument instead of reading
# from the environment variable because during initial
# stages of cloning most of the environment variable
# is not set.
# We need $AD_TOP to update context template adxmlctx.tmp

my $ad_top = $ARGV[1];

# This is the context template
# We will first have to add the custom context variables
# to this file to make oracle's software aware of our
# custom variables.

my $context_template = $ad_top . "/admin/template/adxmlctx.tmp";

# Reference to Array containing references to
# hashes. Each hash contains information about
# custom context variables to be added to the
# context file.
# The first hash represents the
# top level node oa_customized. The rest of the
# context variables will be added under the
# oa_customized node

my $customVar = [
    {
        node => "oa_customized",

```

OAUG Forum at COLLABORATE 06

```

    parent => "oa_context",
  },
  {
    node => "maxconnect",
    attrlist => {
      oa_var => "c_maxconnect",
      scope => "custom",
    },
    value => "20",
    parent => "oa_customized",
  },
  {
    node => "cachesize",
    attrlist => {
      oa_var => "c_cachesize",
      scope => "custom",
    },
    value => "50",
    parent => "oa_customized",
  },
  {
    node => "maxidle",
    attrlist => {
      oa_var => "c_maxidle",
      scope => "custom",
    },
    value => "30",
    parent => "oa_customized",
  },
  {
    node => "englife",
    attrlist => {
      oa_var => "c_englife",
      scope => "custom",
    },
    value => "50",
    parent => "oa_customized",
  },
];

```

```

# Create a new TXK::XML object
# for loading our context file

```

```

my $xmldoc = new TXK::XML;

```

```

# Create a new TXK::XML object
# for loading the context file template
# adxmlctx.tmp

```

```

my $xmldoc_template = new TXK::XML;

```

```

# Initialize the TXK::XML object with our
# context file

```

```

$xmlldoc->loadDocument (
  { file => $context_file }
);

```

OAUG Forum at COLLABORATE 06

```

# Initialize the TXK::XML object with our
# context template

$xmldoc_template->loadDocument (
    { file => $context_template }
);

# Loop through array customVar
# and call routine putCustomNode twice
# for each node to be added. Once for
# adding to the context file and second for
# adding to the context file template.

foreach my $rec (@$customVar)
{
    putCustomNode($xmldoc, $rec);
    putCustomNode($xmldoc_template, $rec);
}

# Write the changed context file to file system
$xmldoc->storeDocument (
    { "file" => $context_file }
);

# Write the changed context file template to the file system
$xmldoc_template->storeDocument (
    { "file" => $context_template }
);

# Routine to add a node
# to the context file or the context template
#
# This routine expects two arguments:
# 1. TXK::XML object for the context file
# 2. Hash reference to the hash with
#    information about the node to be added
# This hash is generally of the following format
# {
#   node => Scalar <Node Name>,
#   attrlist => { Name Value pairs of the attribute list },
#   value    => Scalar < Value for the context variable>
#   parent   => Scalar < Name of the parent node>
# }

sub putCustomNode
{
    my ($xmldoc,$node_to_add) = @_;

    # Check whether the node to be added already exists
    # in the file.
    #
    # This function returns a hash representing the
    # XML node if the node exists. For the details
    # about this hash please refer to the comments in

```

OAUG Forum at COLLABORATE 06

```

# $AU_TOP/perl/TXK/XML.pm

my $temp = $xmldoc->getNode (
    { node => $node_to_add->{"node"} }
);

# If the node does not exist
# add it to the context file
unless ( defined $temp )
{
    $xmldoc->addNode($node_to_add);
}
}

```

If we use the above programmatic method to add our context variables we will not be able to maintain our custom variables through OAM, which is very convenient since OAM automatically regenerates all the context files (in a multi node system) for any change in any custom variables. Moreover, it provides us with a central repository for all our custom variables. We also cannot use OAM to see the history of changes in our context files. This can be easily taken care by updating the existing record for our context file to status 'H' in `fn_d_oam_context_files` table, inserting the changed context file into the table using executable `FNDCPUCF` (in `$FND_TOP/bin`) and inserting a new row into `fn_d_oam_context_custom` for each of our custom variable.

If you have a multi node install, you will have to run the above code on each node to update each of the context files. Moreover if you want to delete a custom node, you still have to use OAM. So there are limitations to the programmatic method I have described so far. At the end of the paper there is a piece of perl code, which takes care of these drawbacks as well. It basically emulates exactly what OAM does when adding or deleting a node.

Customizing an AutoConfig template file delivered by Oracle:

In the report server configuration file example, we added four custom context variables to replace the hard coded values in the configuration file. The second step would be to change the template to replace those hard coded values with our custom context variables.

In older version of autoconfig oracle did not document any method to customize an oracle-supplied template. If we tried to customize AutoConfig generated files by manually editing them, the changes were lost every time AutoConfig ran. To avoid this, oracle some times instructed us to add the customizations between "Begin/End customizations" blocks in the configuration files. This method is quite limited and inflexible. Not to mention it won't work in the above case. Because if we add another `initengine=%c_initengine%` within "Begin/End customizations" at the end of the file, the generated configuration file will end up having two `initengine` entries and report server would fail to startup. In some configuration files the second entry supersedes the first entry, in those cases this method would work fine.

The other unsupported method was to replace the existing template with the custom template. This would work but any autoconfig patch would replace the customized template. We will have to keep customizing the new templates. Moreover, we will have to take a inventory of templates replaced by any autoconfig patch before applying it, to see whether it is replacing our customized template or not. That would be a maintenance nightmare.

Starting with ADX.F Oracle has come up with a fully supported and flexible method of customizing the oracle-supplied templates. This method involves copying the seeded template into `<PROD_TOP>/admin/template/custom` directory.

For example to customize `$AD_TOP/admin/template/REP60_server.ora` we will have to copy the custom template into `$AD_TOP/admin/template/custom/REP60_server.ora`. When autoconfig runs it will instantiate the custom template instead of the seeded template if it finds a custom template. Autoconfig before instantiating the custom template will check the version of the custom
 OAUG Forum at COLLABORATE 06

template. If the version is same as the seeded template than it will instantiate the custom template. If the version is different (higher or lower) than autoconfig will stop executing and inform us about the version mismatch in the log file. At that point we will have to review our customizations and if they are still required we will have to copy the new version of autoconfig template to the custom directory and edit it with our customizations.

So to customize our report server configuration file REP60_server.ora we will copy the file to \$AD_TOP/admin/template/custom/REP60_server.ora and customize this copy. The customized copy will look like the following:

```
mailprofile="$Header: REP60_server.ora 115.6 2004/11/30 05:20:55 ptiwari ship $"
nlssupport=yes
maxconnect=%c_maxconnect%
cachedir="%s_806config_home%/reports60/server/cache"
cachesize=%c_cachesize%
minengine=%s_minengine%
maxengine=%s_maxengine%
initengine=%c_initengine%
maxidle=%c_maxidle%
security=1
englife=%c_englife%
```

The only question that remains is, how do we know REP60_server.ora is the template for reports server configuration file. In this case I found out by “grepping” for the configuration file name in the driver files (mostly adtmpl.drv and fndtmpl.drv since they manage most of the configuration files). The third field in the template’s file entry in the product driver gives us the template name. Before ADX.F this was the only way to find template files for configuration files. With ADX.F, since oracle has decided to completely support autoconfig customizations, they have provided a script \$AD_TOP/bin/adtmplreport.sh to find out the template names. This script in turn runs the java class oracle.apps.ad.tools.configuration.ATTemplateReport. This script is a generic script to generate report on all the autoconfig templates and corresponding configuration files. But it also provides a way to find out the template file from the given configuration file and vice versa. It also reports if there is a custom template for the given configuration file.

```
adtmplreport.sh contextfile=/u01/home/applHOPP/admin/HOPP_jinzo.xml \
target=/u01/app/oracle/product/config/HOPP_jinzo/8.0.6/reports60/server/REP60_HOPP.ora verbose
#####
Generating Report .....
#####
APPL_TOP Context
[AD_TOP]
TEMPLATE FILE : /u01/home/applHOPP/ad/11.5.0/admin/template/REP60_server.ora
TARGET FILE : /u01/app/oracle/product/config/HOPP_jinzo/8.0.6/reports60/server/REP60_HOPP.ora
```

From the above report it is quite clear that our template file is \$AD_TOP/admin/template/REP60_server.ora.

With this method we can pity much customize any oracle-supplied template except for the templates, which are marked as LOCKED in their driver file entry. Autoconfig ignores the custom template for such templates. For e.g. the following entry in adtmpl.drv stops us from customizing the adconfig.txt template.

```
ad admin/template adconfig.txt INSTE8 <s_at>/admin adconfig.txt 600 LOCK
```

Creating custom Autoconfig templates and custom Autoconfig driver:

In the previous section we saw, how to customize oracle-supplied templates. We can even create our own custom templates and custom driver files under custom <PROD_TOP> to manage

OAUG Forum at COLLABORATE 06

custom application configuration files with autoconfig. It mainly involves adding custom product top to the context file, creating the custom autoconfig template directory, developing the custom autoconfig template, adding custom context variables to the context file (if any used in the custom template), creating the custom autoconfig driver directory and finally developing the custom owned driver file. This can be best demonstrated with an example.

Oracle Application's main environment file is APPS<CONTEXT_NAME>.env, which in turn runs <CONTEXTNAME>.env seeded environment file and custom<CONTEXT_NAME>.env if one exists. Let's create autoconfig template and driver file under custom product \$MY_PROD_TOP to maintain this custom environment file.

a) Adding custom product top to the context file:

Autoconfig picks up the driver files to process from the list of products in the context file. So if we want it to process the driver file in \$MY_PROD_TOP, we will have to add this product top to the context file. You can use either OAM or the programmatic approach to add this product top as a context variable. Following are the details for this context variable:

```
name = my_prod_top
oa_var = c_my_prod_top
Default Value = %s_at%/my_prod/11.5.0
Title = My Product top
Description = This is my product top
oa_type = PROD_TOP
```

b) Create the custom owned AutoConfig template directory-

Autoconfig templates as a standard are stored in \$PROD_TOP/admin/template directory. So we will need to create the \$MY_PROD_TOP/admin/template directory to store our custom templates.

c) Develop the custom owned AutoConfig template file:

Following is our custom template.

```
MY_PROD_TOP="%s_at%/myprod/11.5.0"
export MY_PROD_TOP

MY_PROD_TOP1="%s_at%/myprod1/11.5.0"
export MY_PROD_TOP1

MY_PROD_TOP2="%s_at%/myprod2/11.5.0"
export MY_PROD_TOP2
```

The custom template can be named anything, we will name this custom_ux.env (ux stands for UNIX, we can similarly create custom_nt.env and instantiate the appropriate template based on the platform) and this template will be located in \$MY_PROD_TOP/admin/template directory created above. We are using \$APPL_TOP's context variable to build our product tops instead of using new context variables altogether.

d) Adding custom context variables to the context file:

Since we have not used any new context variables in our template above, we don't have to add any variable to the context file.

e) Creating the custom autoconfig driver directory:

Autoconfig driver files as a standard are stored in \$PROD_TOP/admin/driver directory. So we will need to create the \$MY_PROD_TOP/admin/driver directory to store our custom driver file.

f) Developing the custom owned driver file:

We have seen the driver file syntax above. Based on the above syntax our driver file would like like the following:

```
if platform NT
  myprod admin/template custom_nt.env INSTE8 <s_at> custom<s_contextname>.cmd 700
endif
```

```
if platform UNIX
  myprod admin/template custom_ux.env INSTE8 <s_at> custom<s_contextname>.env 700
endif
```

It is stored in \$MY_PROD_TOP/admin/driver and named myprodtmpl.drv as per the standards.

AutoConfig and multimode installation:

Nowadays most Oracle Applications installations have multiple application servers. There could be multiple web servers and multiple concurrent managers. In case of multi node installation each node has its own context file i.e. <CONTEXT_NAME>.xml and its own set of configuration files. These configuration files will be generated from the same set of autoconfig templates since each node is required to be at same level of autoconfig patches.

In the multi node installation we will have to run autoconfig on each node. Each time node specific configuration files are generated, but database updates (i.e. profile option etc) are done each time. So the final value of these options will be the value from the last autoconfig run. This small piece of information should be always kept in mind while running autconfig on multiple node installations.

Troubleshooting AutoConfig:

The amount of processing autoconfig does can be quite overwhelming when debugging problems with autconfig. But autoconfig generates copious logging messages to help us in this problem. Each autoconfig run generates a log file in \$APPL_TOP/admin/<context_name>/log/MMDDhhmm/adconfig.log. This log file has very detailed information about each template autoconfig instantiates and each script it runs.

For each template autoconfig instantiates, it outputs the following in the log file:

```
instantiate file:
  source : <Full path to the template file>
  dest  : <Full path to destination file>
  backup : <original configuration file> to
<APPL_TOP>/admin/<context_name>/out/MMDDhhmm/<original configuration_file>
  setting permissions: 755
  setting ownership: applHOP1:hop1
```

For each script autoconfig runs, it outputs the script name and the output from the script in the log file:

```
/u01/home/comnHOP1/admin/install/HOP1_tristan/txkJavaMailerCfg.sh
```

```
script returned:
*****
```

```
txkJavaMailerCfg.sh started at Sun Jan  8 10:00:04 EST 2006
```

```
The environment settings are as follows ...
```

```
OAUG Forum at COLLABORATE 06
```

```
ORACLE_HOME : <ORACLE_HOME>
ORACLE_SID :
TWO_TASK : <TWO_TASK>
PATH : <PATH>
LD_LIBRARY_PATH : <LD_LIBRARY_PATH>

Executable : /u01/app/oracle/product/8.0.6/bin/sqlplus

SQL*Plus: Release 8.0.6.0.0 - Production on Sun Jan 8 10:00:04 2006

(c) Copyright 1999 Oracle Corporation. All rights reserved.
```

```
Connected.
Executed Successfully
```

```
PL/SQL procedure successfully completed.
```

```
Commit complete.
```

```
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.5.0 - 64bit Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.5.0 - Production
ERRORCODE = 0 ERRORCODE_END
```

```
.end std out.
```

```
.end err out.
```

```
*****
```

With the amount of logging information available, debugging autoconfig problems generally is very straightforward.

If you find that autoconfig has started behaving incorrectly after you have put your customizations in place and you are not sure whether the problem is with the customizations or not, then you can ask autoconfig to ignore customizations and see if the problem goes away or not.

```
<AD_TOP>/bin/adconfig.sh -nocustom contextfile=<CONTEXT> appspass=<APPSpwd>
```

If you find that after running autoconfig, your environment has started having problems then, you can use the restore facility provided by autoconfig. At each run autoconfig backs up the original configuration files in \$APPL_TOP/admin/out/MMDDhhmm directory. It also creates a shell script in this directory called restore.sh. This script restores all your original configuration files. The only catch to this is, restore.sh only restores the configuration files, and updates to the database are not restored. You will have to restore them yourself. One way to achieve this could be to customize the critical script templates to echo the original version of the database options to the autoconfig log file, and then you can restore the original values from the log file. If you are ambitious enough, you can customize each script template to create another version of itself, the only difference being that the new version will restore the original values.

As you can see restoring from autoconfig is not a easy task. We can avoid this problem to a large extent by using autoconfig's test only option. In test only mode autoconfig only generates a difference report between existing configuration files and the new once it would have created if run without test only. In test only mode autoconfig only instantiates the templates, it does not run any script. To run autoconfig in test mode:

OAUG Forum at COLLABORATE 06

```
<AD_TOP>/bin/adconfig.pl contextfile=<CONTEXTFILE_NAME> test
```

Conclusion:

In last twenty pages or so we took a detailed look at autoconfig. Autoconfig has been a blessing for most of the applications dbas since it obviates the need for us the dbas to remember all the files and all the database options which need to be set correctly. But there were some problems since each site is not the same and sometimes autoconfig did not meet the site's requirement, therefore dbas still had to manually update some of these files/options. With fully documented and supported autoconfig customization there is no longer any need for any manual changes.

The only thing I would still like Oracle to do is to fully document the java classes and perl modules they supply so that we can use them effectively to manage and customize autoconfig.

Appendix:

I mentioned in the section of adding custom context variables to the custom file, that I will include a piece of code which emulates the exact behavior of Oracle Application Manager for adding or deleting custom variables. This piece of code can be used for adding or deleting variables from the context file in a single or multi node installation. Even if you don't use this code, reading through this code will be quite informational.

This code does the following:

- Reads the list of context files from applsys.fnd_oam_context_files table where status='S'.

OAUG Forum at COLLABORATE 06

- For each context file:
 - Load the text column into a perl scalar
 - Load the above scalar into a TXK::XML object using TXK::XML::loadstring method.
 - Loops through each custom variable (Stored as hash references in a array reference) to be added and adds them to the TXK::XML object using addNode method.
 - Loop through each custom variable to be deleted (Stored as hash references in a array reference), get the corresponding node using TXK::XML::getNode method and set the "deleted" property of that node to TXK::UTIL::True
 - Update the status to 'H' for the existing record for this context file in applsys.fnd_oam_context_files table.
 - Load the changed context file back into the table using fnd_gsm_util package.
 - Output the TXK::XML object to \$COMN_TOP/temp directory as a temp file.
 - Call fnd_webfile.get_url to insert a record in fnd_file_temp table, with source file as file written in last step, destination file and destination node as selected from fnd_oam_context_files. The url returned by this step is of the format https://<host>;port/OA_CGI/FNDWRR.exe?temp_id=<a number>, where temp_id is the file_id of the row inserted in fnd_file_temp.
 - Use Net::SSLeay::get_https to invoke the url returned in last step. Invoking this URL results in FNDWRR.exe copying the file from source_node:source file to the destination_node:destination file.
- Loops through each context variable to be added, and adds them to applsys.fnd_oam_context_custom table
- Loop through each context variable to be deleted, and deletes them from applsys.fnd_oam_context_custom.

```
#!/u01/home/applHOPP/ad/11.5.0/perl/bin/perl
```

```
BEGIN
```

```
{
  my @inc_array = ("/u01/home/applHOPP/ad/11.5.0/perl/lib/5.8.8/i686-linux",
                  "/u01/home/applHOPP/ad/11.5.0/perl/lib/5.8.8",
                  "/u01/home/applHOPP/ad/11.5.0/perl/lib/site_perl/5.8.8/i686-linux",
                  "/u01/home/applHOPP/ad/11.5.0/perl/lib/site_perl/5.8.8",
                  "/u01/home/applHOPP/ad/11.5.0/perl/lib/site_perl",
                  "/u01/home/applHOPP/au/11.5.0/perl"
                );
  @INC = @inc_array;
}
```

```
use strict;
use DBI;
use DBD::Oracle;
use Sys::Hostname;
use Net::SSLeay qw(&get_https);
```

```
use TXK::XML;
use TXK::Common;
use TXK::Runtime;
```

```
# This will be our staging directory. After adding
# the custom nodes we will store the xml file here
# for
# 1. Reading it again to upload it to fnd_oam_context_files
# 2. Requesting the FNDIFS listner on destination node
```

OAUG Forum at COLLABORATE 06

```

# to overwrite the context file with this one
#
my $staging_dir = $ENV{"COMN_TOP"} . "/temp";

# Reference to Array containing references to
# hashes. Each hash contains information about
# custom context variables to be added to the
# context file.
# The first hash represents the
# top level node oa_customized. The rest of the
# context variables will be added under the
# oa_customized node

my $customAddVar = [
    {
        node    => "maxconnect",
        attrlist => {
            oa_var => "c_maxconnect",
            scope => "custom",
        },
        value    => "20",
        parent    => "oa_customized",
        title    => "Maxium simulatneous Connection Allowed by Reports Server",
    },
    {
        node    => "cachesize",
        attrlist => {
            oa_var => "c_cachesize",
            scope => "custom",
        },
        value    => "50",
        parent    => "oa_customized",
        title    => "Maxium Cache Size",
    },
    {
        node    => "maxidle",
        attrlist => {
            oa_var => "c_maxidle",
            scope => "custom",
        },
        value    => "30",
        parent    => "oa_customized",
        title    => "Maxium idle time allowed before the reports process is killed",
    },
    {
        node    => "englife",
        attrlist => {
            oa_var => "c_englife",
            scope => "custom",
        },
        value    => "50",
        parent    => "oa_customized",
        title    => "Maxium requests a report process is allowed to server",
    },
];

```

Reference to Array containing references to
OAUG Forum at COLLABORATE 06

```

# hashes. Each hash contains information about
# custom context variables to be deleted from the
# context file
# In this case we are deleting the node we are
# adding in this code itself. This obviously
# does not make sense, but this is just for
# demonstration purpose.
my $customDelVar = [
    {
        node => "cachesize",
    }
];

my $appsuser = "apps";

# Apps password is passed as first argument
my $appspass = $ARGV[0];

# Scalars to hold the sql statements and handles to hold the statement handles
# returned after parsing the statements.
my $update_statement;
my $update_stmt_handle;
my $load_statement;
my $load_stmt_handle;
my $upload_ctx_statement;
my $upload_ctx_handle;
my $change_file_statement;
my $change_file_handle;
my $context_custom_sql_handle;
my $context_custom_sql;
my $context_custom_delete_sql;
my $context_custom_delete_handle;

# SQL statement for selecting context file information
# from fnd_oam_context_files
my $select_context_sql = "select path,
    text,
    node_name,
    'adconfig'||round(dbms_random.value(10000,99999))||'.xml' short_name,
    decode(name,'TEMPLATE','TEMPLATE','CONTEXT') name
from fnd_oam_context_files
where (name not in ('METADATA','TEMPLATE') AND
(status is null or upper(status) in ('S','F'))) OR
(name in ('TEMPLATE'))";

# Get the database handle by calling getDBIHandle routine
# We supply the apps user and apps password to this routine
my $dbh = getDBIHandle($appsuser,$appspass);

# Prepare the select statement for selecting information about context files
# from fnd_oam_context_files
my $select_context_sql_handle = $dbh->prepare($select_context_sql,
    {ora_check_sql => 0}
);

$select_context_sql_handle->execute();

```

OAUG Forum at COLLABORATE 06

```

my $array_ref;

# Loop for each context returned by the above select statement
while ( $array_ref = $select_context_sql_handle->fetchrow_arrayref )
{
    # Create a new TXK::XML Object
    my $context_xml = TXK::XML->new();

    # Use loadString method of TXK::XML object
    # to load text returned from select statement
    # into $context_xml object
    $context_xml->loadString($array_ref->[1]);

    # Add oa_customized node to the xml
    # object
    putCustomParentNode ($context_xml);

    # Loop through the custom variable array.
    # Call putCustomNode for each variable
    # to add it to the xml document.
    foreach my $var (@$customAddVar)
    {
        putCustomNode ($context_xml, $var);
    }

    # Loop through the custom variable array.
    # Call delCustomNode for each variable
    # to delete it to the xml document.
    foreach my $var1(@$customDelVar)
    {
        delCustomNode ($context_xml, $var1);
    }

    # Store the changed document in staging directory
    # The name of the document will be
    # adconfig<a random number>.xml
    # We get this temporary name from database itself.
    # The last column short name in the query is for
    # this purpose
    my $staging_file_name = $staging_dir . "/" . $array_ref->[3];
    $context_xml->storeDocument( { file => $staging_file_name } );

    # Update status for existing entry for the context file
    # to 'H' (History)
    # Load the changed context file into fnd_oam_context_files
    loadXML($array_ref->[0], $array_ref->[2], $array_ref->[4], $staging_file_name, $dbh);

    # Invoke fnd_web_file.get_url to change the context file on the file system.
    changeFileSystemXML ($staging_file_name, $array_ref->[0], $array_ref->[2], $dbh);
}

# Insert rows into fnd_oam_context_custom for each
# custom variable added
insertCustom($dbh);

# Delete rows from fnd_oam_context_custom for each
# custom variable to be deleted

```

OAUG Forum at COLLABORATE 06

```

deleteCustom($dbh);

# Destroy the statement handles
$context_custom_sql_handle->finish();
$select_context_sql_handle->finish();
$update_stmt_handle->finish();
$load_stmt_handle->finish();
$change_file_handle->finish();
$upload_ctx_handle->finish();
$context_custom_delete_handle->finish();

# Disconnect from the database.
$dbh->disconnect();

# Routine to Get DBI database handle
# Arguments Passes
# 1. Apps User Name
# 2. Apps Password
# Returns:
# DBI Handle connected using supplied user and password
sub getDBIHandle
{
    my ($user,$pass) = @_ ;
    my $mdbh = DBI->connect(
        "dbi:Oracle:host=mike;sid=HOPP;port=1521",
        $user,
        $pass,
        {
            RaiseError => 1,
            AutoCommit=> 0
        }
    );

    # Since context file is stored as clob
    # We have to unlimite the
    $mdbh->{LongReadLen} = 4581728;
    $mdbh->{LongTruncOk} = 1;
    return $mdbh;
}

# Routine to add oa_customized node the xml document.
# All the custom context variables will be added under
# this node.
# Arguments Passed:
# 1. TXK::XML object for the XML document
sub putCustomParentNode
{
    # Get the TXK::XML object from the argument
    # list
    my ($xmldoc) = @_ ;

    # Check for existence the oa_customized node
    # in the xml document
    my $temp = $xmldoc->getNode ( {"node" => "oa_customized"} );

    # Add node oa_customized if it does not exist
    unless ( defined $temp )

```

OAUG Forum at COLLABORATE 06

```

    {
        my $tmphash = {
            node => "oa_customized",
            parent => "oa_context",
        };
        $xmldoc->addNode($tmphash);
    }
}

# Routine to delete a node from the context file
# or context template
# This routine expects two arguments
# 1. TXK::XML object for the context file
# 2. Hash reference to the hash with
#    information about the node to be deleted
# This hash is of the following format
# {
#     node => "cachesize",
# }
#
sub delCustomNode
{
    my ($xmldoc,$node_to_delete ) = @_;

    my $stemnode = $xmldoc->getNode($node_to_delete);
    $stemnode->{'deleted'} = TXK::Util::TRUE;
}

# Routine to add a node
# to the context file or the context template
#
# This routine expects two arguments:
# 1. TXK::XML object for the context file
# 2. Hash reference to the hash with
#    information about the node to be added
# This hash is generally of the following format
# {
#     node => Scalar <Node Name>,
#     attrlist => { Name Value pairs of the attribute list },
#     value    => Scalar < Value for the context variable>
#     parent   => Scalar < Name of the parent node>
# }

sub putCustomNode
{
    my ($xmldoc,$node_to_add) = @_;

    # Check whether the node to be added already exists
    # in the file.
    #
    # This function returns a hash representing the
    # XML node if the node exists. For the details
    # about this hash please refer to the comments in
    # $AU_TOP/perl/TXK/XML.pm

    my $temp = $xmldoc->getNode (
        { node => $node_to_add->{"node"} }
    )
}

```

OAUG Forum at COLLABORATE 06

```

);

# If the node does not exist
# add it to the context file
unless ( defined $temp )
{
    $xmlDoc->addNode($node_to_add);
}
}

# Routine to insert custom variables into
# fnd_oam_context_custom.
sub insertCustom
{
    my ($dbh) = @_;

    unless ( defined $context_custom_sql_handle )
    {
        # SQL statement for inserting into
        # fnd_oam_context_custom
        $context_custom_sql = "insert into fnd_oam_context_custom
            (name,
             oa_var,
             ctx_type,
             oa_type,
             default_value,
             title,
             description
            )
            values
            (:name,
             :oa_var,
             :ctx_type,
             :oa_type,
             :default_value,
             :title,
             :description
            )";

        # Prepare the insert statement for inserting in
        # fnd_oam_context_custom
        $context_custom_sql_handle = $dbh->prepare($context_custom_sql,
            {ora_check_sql => 0}
        );
    }

    # Loop through the list of custom variables and insert into
    # fnd_oam_context_custom
    #
    # Each hash in the list has the following format
    # {
    #   node    => "maxconnect",
    #   attrlist => {
    #       oa_var => "c_maxconnect",
    #       scope => "custom",
    #   },
    #   value  => "20",
    #   parent => "oa_customized",

```

OAUG Forum at COLLABORATE 06

```

# title => "Maxium simulatneous Connection Allowed by Reports Server",
# }
foreach my $rec (@$customAddVar)
{
    $context_custom_sql_handle->bind_param(":name", $rec->{node});
    $context_custom_sql_handle->bind_param(":oa_var", $rec->{attrlist}->{oa_var});
    $context_custom_sql_handle->bind_param(":ctx_type", "A");
    $context_custom_sql_handle->bind_param(":oa_type", "");
    $context_custom_sql_handle->bind_param(":default_value", $rec->{value});
    $context_custom_sql_handle->bind_param(":title", $rec->{title});
    $context_custom_sql_handle->bind_param(":description", $rec->{title});
    $context_custom_sql_handle->execute();
}
$dbh->commit();
}

# Routine to delete custom variables from
# fnd_oam_context_custom.
sub deleteCustom
{
    my ($dbh) = @_ ;

    unless ( defined $context_custom_delete_handle )
    {
        # SQL statement for inserting into
        # fnd_oam_context_custom
        $context_custom_delete_sql = "delete from fnd_oam_context_custom
            where name = :node_name";

        # Prepare the insert statement for inserting in
        # fnd_oam_context_custom
        $context_custom_delete_handle = $dbh->prepare($context_custom_delete_sql,
            {ora_check_sql => 0}
        );
    }

    # Loop through the list of cutsom variables and delete from
    # fnd_oam_context_custom
    #
    # Each hash in the list has the following format
    # {
    #   node   => "maxconnect",
    # }
    foreach my $rec (@$customDelVar)
    {
        $context_custom_delete_handle->bind_param(":node_name", $rec->{node});
        $context_custom_delete_handle->execute();
    }
    $dbh->commit();
}

# Routine to
# Update status for existing entry for the context file
# to 'H' (History)
# Load the changed context file into fnd_oam_context_files
# Arguments Passes
# 1. Path Name to the context file
OAUG Forum at COLLABORATE 06

```

```

# 2. Staging file name for changed context file
# 3. DBI database handle
sub loadXML
{
    my ($path, $node, $type, $staging_file, $dbh) = @_;

    my ($retcode, $error_message, $upload_retcode, $upload_error_message);

    # We need to update the existing record for the
    # context file from 'S' to 'H' in fnd_oam_context_files.
    # So that when we call fnd_gsm.load_context_file,
    # it creates a new record for instead of updating
    # the existing record with status 'S'.
    # This statement handle is a global variable.
    # It will not be defined only the first time
    # So we will parse this update statement only the first
    # time i.e. parse once execute many.
    unless ( defined $update_stmt_handle )
    {
        $update_statement = "update fnd_oam_context_files
            set status='H'
            where path = ? and
            node_name = ? and
            ( status <> 'H' or status is null )";

        $update_stmt_handle = $dbh->prepare($update_statement);
    }
    $update_stmt_handle->bind_param(1, $path);
    $update_stmt_handle->bind_param(2, $node);

    # We have to call fnd_gsm_util.append_ctx_fragment
    # multiple times for each chunk of the xml document.
    # After consuming the full document we will call
    # fnd_gsm_util.upload_context_file. There is a
    # unless statement here due to same reason as above
    # The arguments passed are
    # 1. document (Document chunk as varchar)
    # 2. Retcode (0 for success >0 for error)
    # 3. Error_message (Error message)
    unless ( defined $load_stmt_handle )
    {
        $load_statement = "begin
            fnd_gsm_util.append_ctx_fragment(?, ?, ?);
            end;";
        $load_stmt_handle = $dbh->prepare($load_statement);
    }
    $load_stmt_handle->bind_param_inout ( 2, \$retcode,50);
    $load_stmt_handle->bind_param_inout ( 3, \$error_message,100);

    # We call fnd_gsm_util.upload_context_file, after
    # calling fnd_gsm_util.append_ctx_fragment multiple
    # times to consume the whole document in chunks
    # The arguments required by this call are
    # 1. Path (This is the full path to context file)
    # 2. Retcode (0 for success >0 for error)
    # 3. Error_message (Error message)
    # 4. context_type (APPS /DATABASE)

```

OAUG Forum at COLLABORATE 06

```

# 5. file_type (CONTEXT/TEMPLATE/METADATA)
unless ( defined $upload_ctx_handle )
{
    $upload_ctx_statement = "begin
        fnd_gsm_util.upload_context_file(?,?,?,?);
    end;";
    $upload_ctx_handle = $dbh->prepare($upload_ctx_statement);
}
$upload_ctx_handle->bind_param ( 1, $path );
$upload_ctx_handle->bind_param_inout ( 2, \ $upload_retcode,50);
$upload_ctx_handle->bind_param_inout ( 3, \ $upload_error_message,100);
$upload_ctx_handle->bind_param ( 4, "APPS" );
$upload_ctx_handle->bind_param ( 5, $type );

$update_stmt_handle->execute();
$dbh->commit();

# Open the staging file which contains the original XML
# document with our added context variables.
# Manipulate it so that @z contains the whole file
# with each character as its array element.
open (IN, "$staging_file" ) ||die ("Could not open $staging_file: $!\n");
my @x = <IN>;
my $y = join "",@x;
my @z = split //, $y;

my $count = 0;
my $str;

# Loop through each character and append each character to $str.
# After every 10000 characters execute fnd_gsm_util.append_ctx_fragment
# with $str as its first argument
foreach my $rec (@z)
{
    ++$count;
    $str .= $rec;
    if ( $count == 10000 )
    {
        $load_stmt_handle->bind_param ( 1, $str );
        $load_stmt_handle->execute();
        $str = "";
        $count = 0;
    }
}
$load_stmt_handle->bind_param ( 1, $str );
$load_stmt_handle->execute();

# Call fnd_gsm_util.upload_context_file
$upload_ctx_handle->execute();
$dbh->commit();
}

sub changeFileSystemXML
{
    my ($staging_file,$dest_file,$dest_node,$dbh) = @_;
    my $url;
    my $host = hostname;

```

OAUG Forum at COLLABORATE 06

```

# fnd_gsm_util.upload_context_file changes the
# node_name in fnd_oam_context_files to the
# current node for TEMPLATE and METADATA
# So we need to supply this host as the
# destination host for these files to fnd_webfile.get_url
if ( $dest_file =~ /adxmlctx/ )
{
    $dest_node = $host;
}

# fnd_webfile.get_url adds a row to the fnd_file_temp
# with all the information we supply to it
# It then returns a url to call FNDWRR.exe with the
# file_id. When we invoking this URL FNDWRR.exe
# copies the source_node:source_file to dest_node:dest_file
# via FNDFS listener on the destination node.
unless ( defined $change_file_handle )
{
    $change_file_statement = "begin
        :url := fnd_webfile.get_url(fnd_webfile.context_file,
            null,
            :gwyuid,
            :two_task,
            :expire_time,
            :source_file,
            :source_node,
            :dest_file,
            :dest_node);
    end;";

    $change_file_handle = $dbh->prepare($change_file_statement);
}
$change_file_handle->bind_param_inout (":url", \$url, 300);
$change_file_handle->bind_param (":gwyuid", "APPLSYSPUB/PUB");
$change_file_handle->bind_param (":two_task", $ENV{"TWO_TASK"});
$change_file_handle->bind_param (":expire_time", 10);
$change_file_handle->bind_param (":source_file", $staging_file);
$change_file_handle->bind_param (":source_node", $host);
$change_file_handle->bind_param (":dest_file", $dest_file);
$change_file_handle->bind_param (":dest_node", $dest_node);

$change_file_handle->execute();
$dbh->commit();

# The url returned by fnd_webfile.get_url is of the following format
# https://<host>:<port>/OA_CGI/FNDWRR.exe?temp_id=<string of numbers>
# If we invoke this URL FNDWRR.exe will copy the file from source_node:source_file
# to dest_node:dest_file (if destination node and destination file is not null).
#
# This copying is done via the FNDFS listener on the destination node.
# If destination node and destination file is null, FNDWRR.exe downloads
# the file to the http client which requested the URL.
#
# We are going to invoke this url via Net::SSLay::get_https
# get_https requires three arguments
# 1. Web Server Host

```

OAUG Forum at COLLABORATE 06

```

# 2. Web Server Port
# 3. Relative URL
# It returns the html output returned by the webserver.
# In case of this URL we expect the output to be "SUCCESS".
#
# So we will have to parse the url returned from fnd_webfile.get_url
# into the above components. I am using the perl regex to do this
# $request_host - Web Server Host
# $request_port - Web Server Port
# $request_url - Relative URL
#
my ( $request_host, $request_port, $request_url, $query_string );
if ( $url =~ /^https:\V([a-z0-9.]*):([0-9]*)\V(VOA_CGIVFNDWRR.exe\?temp_id=[0-9]*)\V/ )
{
    $request_host = $1;
    $request_port = $2;
    $request_url = $3;
}

# Lets invoke get_https with our url
# For this to work FNDIFS should have permission to write
# to the destination directories on the destination nodes
# This permission needs to be at two levels
# 1. OS Level rights to user who starts the FNDIFS listener
# 2. APPLFSWD environment variable in listener.ora
#    should include the destination directory
#
my ($page) = Net::SSLay::get_https ($request_host, $request_port, $request_url);
}

```