XFILES, the APEX 4 version: The truth is in there…

Roel Hartman
Logica
Arnhem, The Netherlands

Keywords:

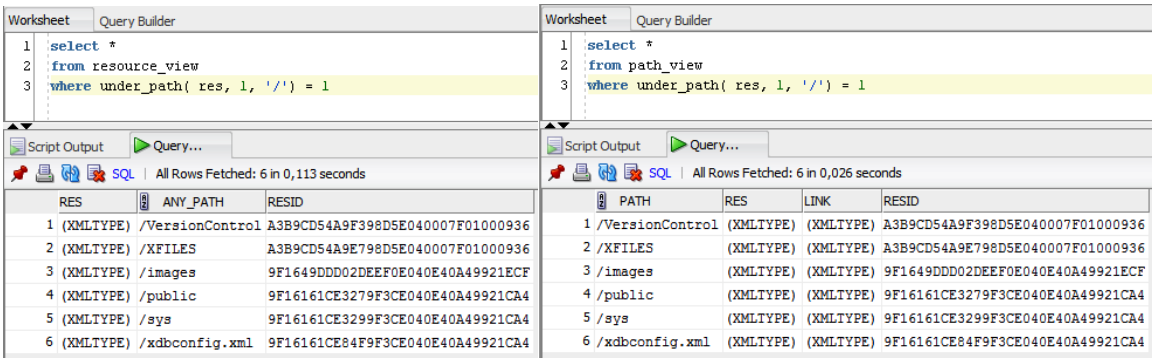
APEX, Oracle Application Express, XML-DB, Version Control, Versioning

Introduction

During Oracle Open World 2008, Mark Drake (senior Product Manager Oracle XMl-DB) and Carl Backstrom (senior Member of Technical Staff Oracle Application Express) did a joint presentation on the combination of XML-DB and APEX. The APEX part was just a first rough cut front end on the XML foundation Mark created. Due to a tragic car accident Carl passed away just a few weeks after this presentation and thus his code was never finished.
Early 2011, Marco Gralike (Oracle ACE Director at AMIS, specialized in XML-DB) and Roel Hartman (Oracle ACE Director at Logica, specialized in APEX), joint forces to take the old XFILES application and recreated an APEX 4 version. Next to that, they also build additional features, so the XFILES application can be used as a version control system for APEX applications: Version Control as a 100% database centric solution!
BTW, Mark Drake is still developing 'his' version of the XFILES. He is now on version 5 and that version is included in the latest virtual box development environments. To reduce any confusing, we (Marco and Roel), renamed the version this paper is about to: XACE – XFILES, APEX Community Edition.

What is XML-DB?

Since version 9.2 of the Oracle Database, XML-DB is available as a no-cost option. XML-DB can handle XML, like storing, consuming, generating and validating. It contains a XDB Repository, which is accessible using http(s), ftp or WebDav. XML-DB supports all XML (and related) standards, like XPath, XSLT, XQuery and XBRL and supports security (using Access Control Lists, ACL), the use of events and...versioning.
The versioning functionality, including check in and checkout functionality is implemented in the DBMS_XDB_VERSION package. All resource (file, folder) manipulation is done using the DBMS_XDB package.
For accessing the contents of XDB Repository two views are at your disposal: PATH_VIEW and RESOURCE_VIEW. PATH_VIEW contains a row for every path to a resource (for instance when using links, you can have more path to the same resource), while RESOURCE_VIEW contains a row for every resource. So PATH_VIEW always contains equal or more rows than RESOURCE_VIEW.
For performance reasons, when selecting from these views, you should use the special SQL functions UNDER_PATH and EQUALS_PATH.
So in order to select the files and folders that are directly under the root folder, you can issue the SQL command as you can see in Illustration 1.



Worksheet | Query Builder

```
select *
from resource_view
where under_path( res, 1, '/') = 1
```

Script Output | Query... | SQL | All Rows Fetched: 6 in 0,113 seconds

	RES	ANY_PATH	RESID
1	(XMLTYPE)	/VersionControl	A3B9CD54A9F398D5E040007F01000936
2	(XMLTYPE)	/XFILES	A3B9CD54A9E798D5E040007F01000936
3	(XMLTYPE)	/images	9F1649DDD02DEEF0E040E40A49921ECF
4	(XMLTYPE)	/public	9F16161CE3279F3CE040E40A49921CA4
5	(XMLTYPE)	/sys	9F16161CE3299F3CE040E40A49921CA4
6	(XMLTYPE)	/xdbconfig.xml	9F16161CE84F9F3CE040E40A49921CA4

Worksheet | Query Builder

```
select *
from path_view
where under_path( res, 1, '/') = 1
```

Script Output | Query... | SQL | All Rows Fetched: 6 in 0,026 seconds

	PATH	RES	LINK	RESID
1	/VersionControl	(XMLTYPE)	(XMLTYPE)	A3B9CD54A9F398D5E040007F01000936
2	/XFILES	(XMLTYPE)	(XMLTYPE)	A3B9CD54A9E798D5E040007F01000936
3	/images	(XMLTYPE)	(XMLTYPE)	9F1649DDD02DEEF0E040E40A49921ECF
4	/public	(XMLTYPE)	(XMLTYPE)	9F16161CE3279F3CE040E40A49921CA4
5	/sys	(XMLTYPE)	(XMLTYPE)	9F16161CE3299F3CE040E40A49921CA4
6	/xdbconfig.xml	(XMLTYPE)	(XMLTYPE)	9F16161CE84F9F3CE040E40A49921CA4

Illustration. 1: SQL statements to select contents from the XML-DB views

Version Control procedure in XML-DB

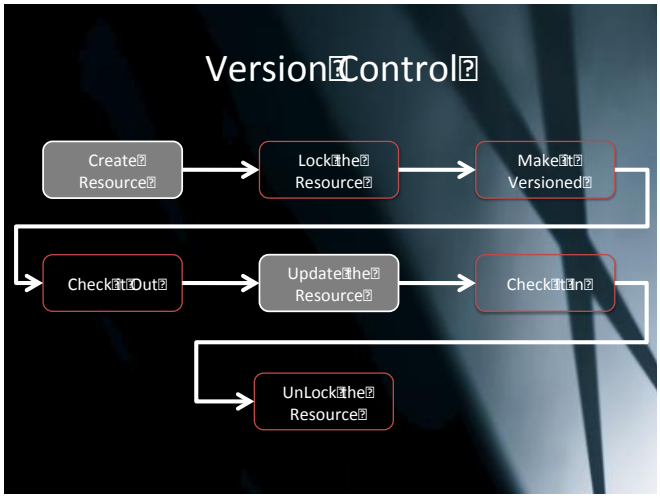


Illustration. 2: The version control process

The version control procedure in XML-DB consists of 7 consecutive steps, as you can see in Illustration 2. The first thing you need ofcourse is something to version, so you first need a resource. Then, before you do anything with this resource, you have to lock it and make it a versioned resource. This step turns a regular resource whose path name is given into a version-controlled resource. This new resource is then put under version control. All other path names continue to refer to the original resource. Then you have to check out the resource, do your updates and check it back in again. Once you're done, you can unlock the resource.

In code you get something like in the next listing.

```
declare
  targetpath varchar2 ( 256 ) := '/public/testcase.txt';
  result     boolean;
  resid dbms_xdb_version.resid_type;
  token varchar2 ( 4000 ) ;
begin
  result := dbms_xdb.createresource
            ( targetpath, 'This is the original content' ) ;
  result := dbms_xdb.lockresource ( targetpath, false, false ) ;
  resid  := dbms_xdb_version.makeversioned ( targetpath ) ;
  dbms_xdb_version.checkout ( targetpath ) ;

  update resource_view
  set res = updatexml ( res
                      , '/Resource/Contents/text/text()'
```

```
                      , 'This is the new content' )
  where equals_path ( res, targetpath ) = 1;

  resid := dbms_xdb_version.checkin ( targetpath ) ;
  dbms_xdb.getlocktoken ( targetpath, token ) ;
  result := dbms_xdb.unlockresource ( targetpath, token ) ;

end;
```

Retrieving old versions

Putting resource under version control would be quite useless, unless you can't retrieve your older versions of a resource. In order to retrieve a list of historic versions of a resource within a SQL statement, we have to create a helper, pipelined, function. See the listing of such a function below (original code by Mark Drake).

```
create or replace
  function getversionhistory (
      path varchar2 )
    return xmlsequencetype pipelined
  as
    resource_id raw ( 16 ) ;
    res xmltype;
    source_list dbms_xdb_version.resid_list_type;
  begin

    select res
    , resid
    into res
    , resource_id
    from resource_view
    where equals_path ( res, path ) = 1;

    pipe row ( res ) ;
    source_list := dbms_xdb_version.getpredecessors ( path ) ;

    while source_list.count > 0
    loop
      pipe row ( dbms_xdb_version.getresourcebyresid (source_list (1))) ;
      source_list := dbms_xdb_version.getpredsbyresid (source_list (1)) ;
    end loop;
    return;
  end;
```

After creating this function, we can issue the following SQL statement:

```
select det.*
from table ( getversionhistory ( '/public/testcase.txt' ) ) vh,
     xmltable
     (xmlnamespaces(default 'http://xmlns.oracle.com/xdb/XDBResource.xsd' )
      , '/Resource' passing value ( vh )
        columns
        version_no number ( 3 ) path '@VersionID',
        date_created timestamp ( 6 ) path 'CreationDate',
        date_modified timestamp ( 6 ) path 'ModificationDate',
```

```
        content clob path 'Contents/text/text()'
      ) det
order by version_no
/
```

And this will result in

```
VERSION_NO DATE_CREATED      DATE_MODIFIED     CONTENT
---------- ----------------- ----------------- ------------------------------
1          09-09-11 10:42:29 09-09-11 10:42:29 This is the original content
2          09-09-11 10:42:29 09-09-11 10:42:31 This is the new content
```

So now we've managed to access the XML-DB content and history within SQL, we can use this technique within APEX as well.

Retrieving old versions

To create a nice user interface on top of the XML-DB structure, we start with a view that will show all the folders:

```
CREATE OR REPLACE FORCE VIEW  "XDB_FOLDERS" ("PATH", "PARENT_PATH") AS
  SELECT any_path                                          AS path ,
    NVL(SUBSTR(any_path, 1, instr(any_path,'/',-1)-1),'/') AS parent_path
  FROM RESOURCE_VIEW
  WHERE UNDER_PATH(RES,'/')                                =1
  AND existsnode(res, '/Resource[@Container="true"]' ) = 1
  UNION ALL
    ( SELECT '/' PATH ,NULL AS PARENT_PATH FROM DUAL
    )
/
```

Using this view, we create a Tree region within an APEX page, using this SQL statement:

```
select case when connect_by_isleaf = 1 then 0
            when level = 1             then 1
            else                            -1
       end as status,
       level,
       substr(PATH,instr(path,'/',-1)+1) as title,
       null as icon,
       '\\'||PATH as value,
       null as tooltip,
'javascript:$s( "P1_FOLDER", "'||PATH||'")'  as link
from
(
select path
,      parent_path
from XDB_FOLDERS
)
start with path  = '/'||:P1_FOLDER
connect by prior path=parent_path
order siblings by upper(path)
```

This will result in a nice graphical representation of the folders in the XDB Repository, like you can see in Illustration 3 below.

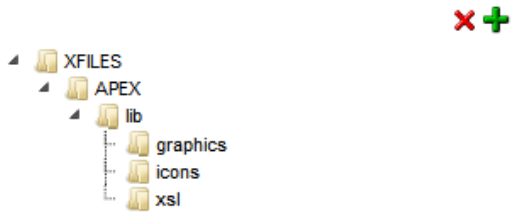
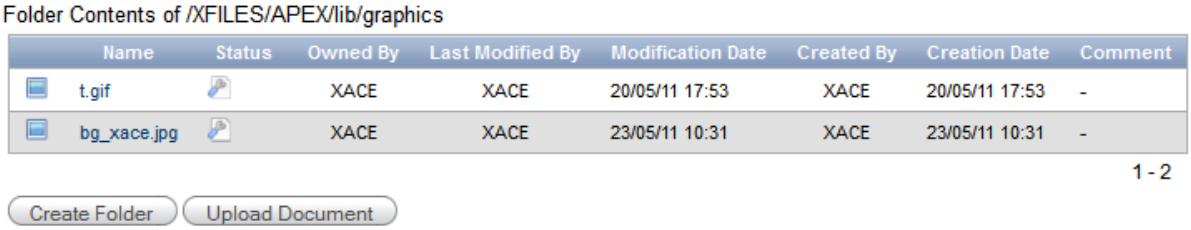
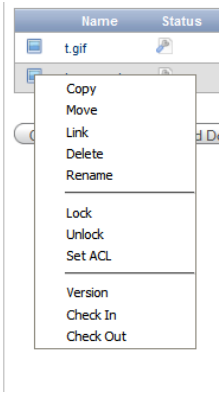


Illustration. 3: APEX Tree view of XML DB Repository

In another region we can show the contents of a folder, using a similar SQL statement (all code is available for download, so if you're interested, download the source and inspect it – see the last paragraph for details).





Using other standard DBMS_XDB functionality for creating folders, uploading documents, copying, moving, creating links, deleting, renaming, lock and unlock, versioning and checkin/out are also exposed via the APEX GUI using a right-click popup menu as you can see in the illustration to the left.
You can also use these features to easily manage the contents of your images directory for APEX itself (when running the EPG – another XML-DB feature).

Version Control in APEX – the XACE application

APEX itself doesn't have a version control feature. Just like regular PL/SQL development, all pages and other components can be exported and stored in your favourite version control tool. For exporting pages you can use the external ApexExport utility, export pages from the builder or use the SQL Developer.
But with all the knowledge we gained from the previous examples, we can also use the XML-DB Repository to store the APEX export (sql) files. So with some additional coding on top of the previous example and the use of the APEX Views, we get a version control system for APEX within the database itself! From this front end you can, with just one click (or even with a regular interval using DBMS_SCHEDULER) create a new version of your page export in the Repository. And of course, you can also retrieve previous versions and then re-install these.

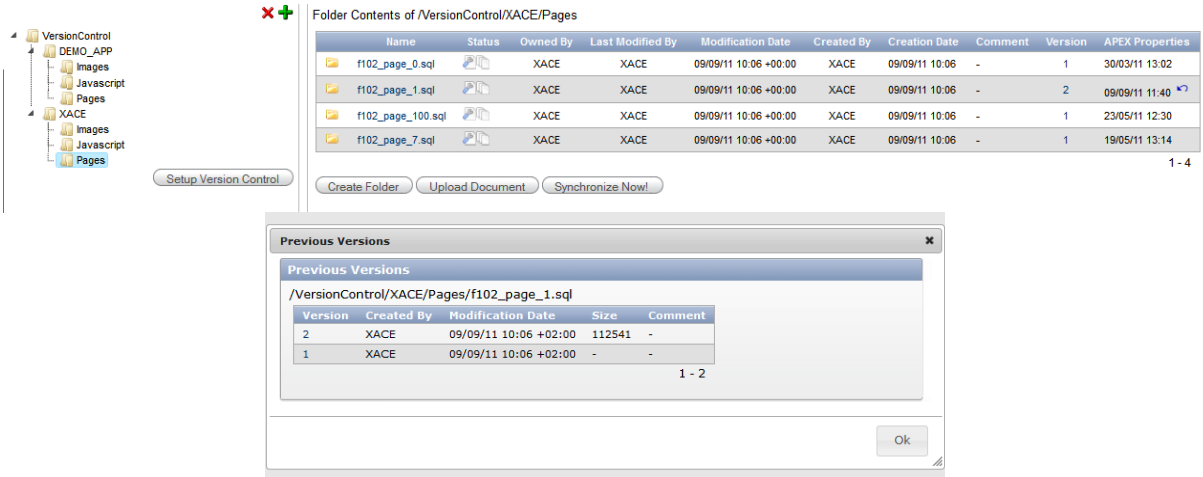


Illustration. 4: APEX Version Control build in APEX and XML-DB

Additional add-on

Wouldn't it be nice when this Version Control application was directly and easily accessible from the APEX Builder itself? You can do that by using the System Message feature of APEX. So to get a (nice) icon in the APEX Builder pointing the the XACE application, login as ADMIN and go to Manage Instance > Define System Message. In the box presented there enter this piece of Javascript:

```
<script type="text/javascript">
$(function(){
if ( $v('pFlowStepId')=='1000')
{
  // Create Image + Link to Version Control on this page"
$('.apex-list-horizontal').append('<div class="noncurrent"><div
class="image"><a title="Version Control"
href="f?p=XACE:1::VC:::P1_FOLDER:VersionControl" target="_blank"><img
width="128" height="128" alt="Version Control" title="Version Control"
src="/i/apex/builder/menu-ws-manage-128.gif"></a></div><div
class="label"><a title="Version Control"
href="f?p=XACE:1::VC:::P1_FOLDER:VersionControl" target="_blank">Version
Control</a></div></div>');
}
});
</script>
```

Now you get an additional icon on your APEX Builder Home screen, and clicking on it will open up the XACE application in a separate window.

Disclaimer: This works in APEX 4.0, in 4.1 the javascript would probably be different!

Download available!

A fully, self installing export of this APEX application is available on SourceForge. Feel free to download it, install and play around. You area also invited to add your own additional functionality to this open source project and share it with your peers.

Contact address:

Roel Hartman
Logica
Meander 901
P.O. Box 7015
6801 HA ARNHEM
The Netherlands

Phone:	+31(0)6-29543729
Email	roel.hartman@logica.com
Internet:	www.logica.com
Blog:	roelhartman.blogspot.com