The OWconfig Access Method

March, 1994, Rev. 1.4

John Saare

Introduction:

The OWconfig Access Method was developed to standardize orderly access to and manipulation of an OWconfig database file. The OWConfig API has been defined to satisfy the requirements of the Xsun server and the kdmconfig utility. It is also used by the DDK utilities “OWmerge” and “OWremove” as the means by which IHVs may (de)install the configuration information of their DDX packages.

The OWConfig API very closely resembles an internal interface already defined in the Xsun server. This resemblance will hopefully avoid untimely destabilization of the Xsun server.

The requirements of the “kdmconfig” utility being developed by SSS are the driving forces behind the new functionality in the OWconfig Access Method. The XSun server requires R/O access to an OWconfig database file. The “kdmconfig” utility requires R/W access to an OWconfig file. Since a R/O implementation shares a substantial amount of functionality with a R/W implementation, it was decided to develop a standard API.

Once a standardized means of manipulating an OWconfig database was developed, it was appropriate to bring the benefits of this standardization to the IHV de(installation) procedure. The DDK utilities “owmerge” and “owremove” were developed for that purpose, and themselves use the OWconfig Access Method.

The OWConfig Database:

An OWConfig database is a hierarchy of lists of name/value pairs. The meaning of a particular name/value pair depends upon its position in the hierarchy, as well as the application’s interpretation of its value. The access method implements each node of the hierarchy with two primary parts: its relationship to other nodes, and a name/value pair. The “attribute” node is the only node whose “value” is exposed directly via the API; the “value” of other node types is significant only to the access method implementation. More concretely:

An OWConfig database is a list of “classes”; each “class” has a name.
A “class” is a list of “instances”; each “instance” has a name.
An “instance” is a list of “attributes”; each “attribute” has a name and a value.

As an example, a typical OWConfig database file will contain a declaration of an instance of class “XDISPLAY” whose name will be “0” (for screen 0). This instance of the “XDISPLAY” class may contain definitions for attributes such as “coreKeyboard” and “corePointer”. The OWconfig file may contain several declarations of instances of class “XDISPLAY”.

The access method does not enforce class/instance/attribute naming conventions, nor does it check values of attributes.

For a more complete description of the syntax and content of an OWconfig file, please refer to Appendix A, “The OWconfig File”, of the document entitled: “OpenWindows 3.4 Server Device Developer’s Guide”.

The OWConfig API:

The C language definition of the OWConfig Access Method API may be found in the include file “/usr/openwin/include/X11/Sunowconfig.h”.

int
OWConfigInit(char *readfile1, char *readfile2, char *writefile, char *(*allocmem)(), void(*freemem())

readfile1: an OWconfig filepath.
readfile2: an OWconfig filepath.
writefile: Pathname of file to write resultant database to during OWConfigClose.
allocmem: function pointer to malloc-like allocate routine.
freemem: function pointer to free-like allocate routine.

Return:

OWCFG_OK: Operation completed successfully.
OWCFG_OPEN1FAIL: Unable to open readfile1.
OWCFG_OPEN2FAIL: Unable to open readfile2.
OWCFG_LOCK1FAIL: Unable to read lock readfile1.
OWCFG_LOCK2FAIL: Unable to read lock readfile2.
OWCFG_SYNTAX1: Syntax error in readfile1.
OWCFG_SYNTAX2: Syntax error in readfile2.
OWCFG_ALLOC: Ran out of memory..
Note: If readfile1 and readfile2 are both non-NULL parameters, and the read of readfile1 succeeds but the read of readfile2 fails, OWCFG_OK will be returned.

OWConfigInit will read the OWconfig files named by readfile1 and readfile2. A NULL pointer passed through either readfile1 or readfile2 indicates that no operation should be performed using the NULL parameter. Class instances and attribute/value pairs read from readfile2 will override those read from readfile1. The replacement will occur on a per attribute/value pair basis. Before a database file is read, it is read locked using the fcntl facility. Immediately after the file is read, it is unlocked. Allocmem defines the memory allocator for use by this invocation of OWconfig. Freemem defines the memory free routine for use by this invocation of OWconfig.

int
OWConfigValidate(void)

Return

OWCFG_OK: Operation completed successfully.
OWCFG_FSTAT1FAIL: Attempt to check the timestamp of readfile1 failed.
OWCFG_FSTAT2FAIL: Attempt to check the timestamp of readfile2 failed.
Note: Additionally, all return values from OWConfigInit are possible..

Checks the validity of the internal database by noting the last modified time stamp on the OWconfig files. If the internal database is out of date, the OWconfig files are reread. This feature has been

provided for the use of the OpenWindows XINPUT server extension.

char **
OWConfigGetClassNames(char *class)
class: name of class for which to name all instances.
Return:
(char **) to list of class instance names, or NULL if class did not exist.
OWConfigGetClassNames returns a list of the names of all the instances of the named class. The end of the list is indicated by a NULL pointer. All users of this function should call OWConfig-FreeClassNames to free the list and the strings it points to.

void
OWConfigFreeClassNames(char **list)
list: NULL terminated list of strings to free.
Frees results of OWConfigGetClassNames.

char *
OWConfigGetAttribute(char *class, char *name, char *attribute)
class: name of class to which named attribute belongs.
name: name of instance of class to which named attribute belongs.
attribute: name of sought attribute.
Return:
(char *) to value of attribute or NULL if attribute could not be found.
OWConfigGetAttribute returns the string value of the requested attribute. The string returned by this function may be freed using OWConfigFreeAttribute.

void
OWConfigFreeAttribute(char *attribute)
attribute: string, allocated by OWConfigGetAttribute, to be freed.
Frees string returned by OWConfigGetAttribute.

int
OWConfigSetInstance(char *class, char *name, OWconfigAttributePtr attr, int numberIn-Attr)
class: name of class into which listed attributes will be placed.
name: name of instance of class into which listed attributes will be placed.
attr: list of attributes.
numberInAttr: number of entries in list of attributes.
typedef struct {char *attribute, *value;}OWConfigAttribute, *OWConfigAttributePtr;
Return:
OWCFG_OK: Operation completed successfully.
OWCFG_ARGS: class, name, and attr must all be non-null.
Note: Additionally, all return values from OWConfigSetAttribute are possible..
OWConfigSetInstance is a convenient front end to OWConfigSetAttribute. It takes a list of attributes and adds them to an instance of a class. If that class and/or instance does not already exist, then it is or they are created. If that instance does exist, replacements, when necessary occur

on a per attribute/value pair basis, otherwise, they are merely added to the instance. The definition for OWConfigAttributePtr may be found in the include file “sunowconfig.h”.

OWConfigAttributePtr
OWConfigGetInstance(char *class, char *name, int *numberInAttr)

class: name of class from which to list attributes.
name: name of instance of class from which to list attributes.
numberInAttr: receives number of attributes in returned list.

Return:

OWConfigAttributePtr or NULL.

OWConfigGetInstance returns a list of attribute definitions. Use OWConfigFreeInstance to free the memory allocated to the information returned by OWConfigGetInstance.

void
OWConfigFreeInstance(OWconfigAttributePtr attr, int numberInAttr)

attr: list of attributes to free.
numberInAttr: number of attributes in list.

Frees a list created by OWConfigGetInstance.

int
OWConfigRemoveInstance(char *class, char *name)

class: class name of instance to remove.
name: name of instance to remove.

Return:

OWCFG_OK: Operation completed successfully.
OWCFG_ARGS: class and name must both be non-NULL.
OWCFG_INSTANCENAME: the instance named by class and name did not exist.

OWConfigRemoveInstance removes, from the internal database, an instance of a class, identified by “class” and “name”.

int
OWConfigSetAttribute(char *class, char *name, char *attribute, char *value)

class: name of class into which is placed the attribute/value pair.
name: name of instance of class into which is placed the attribute/value pair.
attribute: name of the attribute.
value: value of the attribute.

Return:

OWCFG_OK: Operation completed successfully.
OWCFG_ARGS: All parameters must be non-NULL.
OWCFG_ALLOC: Ran out of memory.

OWConfigSetAttribute takes an attribute/value pair and adds it to an instance of a class. If that class and/or instance does not already exist, they are or it is created. If the named attribute already exists within the instance, it is replaced.

int
OWConfigClose()
Return:
OWCFG_OK: Operation completed successfully.
OWCFG_OPENTMPFAIL:
OWCFG_OPENWFAIL:
OWCFG_LOCKWFAIL:
OWCFG_RENAMEFAIL:
If the value specified by OWConfigInit for writefile was NULL, nothing is written and the database is freed. If a writefile was specified, the database is written out in "OWconfig" syntax and the internal database is freed.
When a file is to be written, the existing target file, if any, is write locked using fcntl. If the lock succeeds, the new OWconfig file is written to the same directory, with a temporary name (the temporary name is derived from the PID of the locking process). The old file is removed and the new file is renamed to match the old file. Comments are retained in a fashion that should suffice in most instances. Read the section entitled "Retained Comments" for more detail.

void
OWConfigSetPackage(char *package)
package: name of package with which to associate subsequent instances.
This function establishes the name of the package to associate with database resources created by subsequent calls to OWConfigSetAttribute or OWConfigCreateClass. If OWConfigSetPackage is never called, a value of "DEFAULT" is assumed.

int
OWConfigRemovePackage(char *package)
package: name of package associated with resources to be removed.
Return:
OWCFG_OK: Operation completed successfully.
OWCFG_ARGS: package must be non_NULL.
OWCFG_PKGNAME: Package not found.
This function removes any resources from the database that are associated with the named package.

OWConfig Access Method Packaging:
The API components of the access method will be placed in the following locations:
/usr/openwin/lib/libowconfig.so.0
/usr/openwin/lib/libowconfig.so
/usr/openwin/lib/libowconfig.a
/usr/openwin/include/X11/Sunowconfig.h

Retained Comments:
The OWconfig Access Method will be used by the owmerge and owremove utilities, as well as the kdmconfig utility, to edit an OWconfig database file. The means by which this was accom-

plished in the past, by IHV installation and de-installation scripts, was ad hoc use of special shell script magic that was sensitive to a particular package's resource delimiting scheme that was implemented with various forms of comments.
To illustrate the problem:

- OW3.3 is installed.
- IHV OW3.3 package is installed.
- OW3.4 is installed, as an upgrade to OW3.3.
- The IHV eventually distributes an OW3.4 package.
- The IHV OW3.4 package is installed as an upgrade

If the IHV's de-installation process is to continue functioning after OW3.4 is installed, the comments that the IHV's OW3.3 package is depending upon to identify itself must remain intact. It is also possible that the customer wishes to install an IHV's OW3.3 package AFTER OW3.4 installation, cuz that's the only one they've got.
The access method should at least make an attempt to preserve comments in a fashion sufficient to bridge the period of time in which both forms of OWconfig manipulation must coexist. The following describes one possible means of doing so.
As an OWconfig file is read, instances are associated with a package. If a comment is encountered prior to an instance, all instances preceding the next "package" statement are identified as "old style" instances. Comments preceding each instance will be "attached" to the instance, i.e., "old style" instances MAY be prefixed by comments. If comments are encountered that do not precede an "old style" instance, they will be attached as a suffix to the last "old style" instance in the file, e.g., trailing comments in an OWconfig file. If a "package" statement is encountered prior to an instance, all instances following that package statement are associated with the named package, until either another package statement or comment is encountered. Instances that are explicitly associated with packages do NOT retain comments.

```
#All instances declared following this
#comment, but preceding the next are "new style"
#declarations, i.e., THIS comment block will
#NOT be retained.
package="ZOOTCO"
class="zootClass" name="ZootName"
zootAttr="My name is Zoot";
#BEGIN A ZootCo Production...
#All instances following THIS comment block
#are "old style" because they follow this comment
#block. This comment block will be associated
#with the instance declaration that immediately
#follows it, i.e., this comment block will be retained.
class="oldClass" name="oldName"
oldAttr="This instance is prefixed by a comment";
class="oldClass" name="lastOldName"
oldAttr="This instance is suffixed by a comment";
#This comment block is associated with the instance
```

```
#declaration that immediately precedes it, i.e., this comment
#block will be retained.
#END ZootCo
```

Comment retention is only significant when the access method is used in its R/W capacity, i.e., when the access method is used in a R/O capacity, comments are ignored.

OWConfig Utilities:

The following utilities may be used by IHV package installation/de-installation scripts, to add or remove their package's configuration information.

owmerge <packageDatabaseFile> <targetDatabaseFile>

The package database file identified by <packageDatabasFile> is merged with the database file identified by <targetDatabaseFile> and the result is written back to <targetDatabaseFile>.

owremove <packagename> <targetDatabaseFile>

All database resources associated with <packagename> are removed from the database file identified by <targetDatabaseFile>.

OWConfig Utility Packaging:

The OWConfig utilities will be placed in the following locations:

/usr/openwin/bin/owmerge
/usr/openwin/bin/owremove