External application interface in AlphaTcl

Frédéric Boulanger

June 24, 2003

Abstract

This file constitutes the literate programming source for the xserv extension of AlphaTcl. This extension allows the definition of interfaces for external services, and the declaration of implementations of these services with other applications.

Contents

Conventions used in this paper

In syntax descriptions, a `typewriter` font is used for explicit text. A named syntactic unit is written as ⟨*unit*⟩. In the special but very common case that the syntactic unit is precisely a word for Tcl, it is instead written as {*word*}, i.e., with braces instead of angle brackets. Optional and repeated elements in syntax descriptions are denoted as in regular expressions, using question marks, asterisks, and plus signs, e.g.

`set` {*var-name*} {*value*}$^{?}$
`list` {*item*}*
`append` {*var-name*} {*string*}$^{+}$

Parentheses can be used to group syntax elements, e.g.

`return` $\left(\texttt{-code}\ \{\textit{code}\}\right)^{?}$ {*string*}

The same conventions are used for specifying the structure of lists.

1 Introduction

The xserv extension provides the minimal features that allow AlphaTcl to drive external applications through well defined interfaces. Each interface defines a standard way to ask for a service, no matter which application is used to provide it. Since these services are provided by applications other than *Alpha*, we call them “external services” or XSERVs.

An XSERV describes a service which may depend on parameters. For instance, we can consider a `openURL` service which displays an URL. Its behavior depends obviously on the URL we want to display, so this service will have a `url` parameter.

There are generally several means to provide the service described by an XSERV: we can use any browser to display an URL. However, each browser has its own syntax for performing the requested action.

For each possible implementation of a service, we need to describe how to build the request for the application from the parameters of the XSERV. This task is handled be a “driver” script and depends on the invocation mode of the implementation. The current version of xserv supports the following implementation modes:

App a Mac OS application is launched, and the driver communicates with it by Apple events to provide the service;

Shell a program is launched with the output of the driver as its standard input;

InSh a program is launched, and the output of the driver is written to its standard input. A window displays the output of the program and acts as a console. The interactive run of the program in the window relies on the `InSh` *Alpha* mode;

Exec the driver returns a command line which is executed by the `exec` Tcl command;

Alpha the driver uses AlphaTcl commands to provide the service (not really an external service, but sometime useful).

Once an XSERV has been declared and implementations of this service have been registered, we should be able to choose an implementation and to invoke the service. The choice of an implementation may be driven by personal taste or by the availability of some applications on a given platform.

2 Usage

2.1 Basic functionalities

`xserv::declare` (proc) An XSERV is declared with :

`xserv::declare` {*xserv name*} {*description*} {*param*}*

where {*xserv name*} is the name of the new service, {*description*} is a textual description of the role of the service, and each {*param*} is either the name of a parameter or a list containing the name of a parameter and its default value.

`xserv::declare` returns 0 if it fails to declare the service (for instance if the service is already declared), and 1 if the service declaration was successful.

For instance, the following code:

```
xserv::declare viewURL {Display an URL} url
```

declares an XSERV named `viewURL` which takes a `url` parameter and provides a service described by the sentence "`Display an URL`"; while:

```
xserv::declare TeX "Typeset a file with TeX" file {format latex}
```

declares an XSERV named `TeX` which takes a `file` and a `format` and provides a service described by the sentence "`Typeset a file with TeX`". If the `format` parameter is not given a value when the service is invoked, `latex` will be used as its default value.

`xserv::register` (proc) Once a service has been declared, we can register applications that implement this service with :

`xserv::register` {*xserv name*} {*impl name*} ({*key*} {*value*})$^+$

{*xserv name*} is the name of the service which is implemented by the application;

{*impl name*} is the name of the implementation of the service. This is a symbolic name which is only meant to be used by human beings and may be different from the real program name;

The list of key-value pairs defines how this implementation provides the service. Any key is allowed, but the current version interprets only the following keys:

-sig the associated value is the signature, or creator code, of the Mac OS application used to provide the service;

-path the associated value is the path to the program used to provide the service;

-shell the associated value is the name of the program which will receive the value returned by the driver script on its standard input;

-driver the associated value is the Tcl script which will drive the external program according to the parameters. This script will find the value of the parameters in the `params` array. The `params` name is the only "reserved" name in the context of the driver script. If the XSERV has a `file` parameter, the driver script will find the value of this parameter in `$params(file)`.

The `xservTarget` parameter is added to the parameter array in the `App` invocation mode. Its value is the target process to which Apple events should be sent.

The `xservInteraction` parameter is added to the parameter array in all invocation modes. Its value is `0` when the user should not interact with the application (background invocation) and `1` when the user should be able to interact with the application (foreground invocation).

All parameter names beginning with `xserv` are reserved for future extensions of xserv.

For the `Exec` and `Shell` invocation modes, the driver should return a list of words. xserv will take care of escaping spaces within these words so that the shell or the exec command receive a command line containing the words of the list.

For the `App` and `Alpha` invocation modes, the driver should execute the AlphaTcl commands required to obtain the desired result.

-mode the associated value is the invocation mode of the implementation, as seen above. It may be omitted, in which case it will be set to `App` if the `sig` key is present, to `Shell` if the `shell` key is present, and to `Exec` in the other cases.

-progs the associated value is a list of command-line programs which are needed by the driver script. When using this implementation of the service, `xserv` will find where these programs are, asking the user if several copies are available or if no such program can be found in the command path. For each program named {*prog*} in this list, a variable named {*xserv-prog*} will be available to the driver script, and its value will be the full path to the {*prog*} program to use.

To register Internet Explorer as an implementation of the `viewURL` service on Mac OS, we could write:

```
xserv::register viewURL Explorer -sig MSIE -driver {
  AEBuild $params(xservTarget) WWW! OURL \
   ---- [tclAE::build::TEXT $params(url)]
}
```

Internet Explorer is identified by its `MSIE` creator code. The script uses the `OURL` Apple Event from the `WWW!` suite to ask Internet Explorer to display the URL. `$params(url)` will expand to the value of the `url` parameter of the XSERV when it is invoked. `$params(xservTarget)` will expand to the name of the Internet Explorer application. This application will be found by asking the system for an application with creator code `MSIE`, or by asking the user to locate it if the system cannot find it.

Of course, it is possible to register several applications for one service. If we want to use the Mac OS Help Viewer for `viewURL` (this will work only for `file::` URLs), we can write:

```
xserv::register viewURL AppleHelp -sig hbwr -driver {
  AEBuild $params(xservTarget) GURL GURL \
   ---- [tclAE::build::TEXT $params(url)]
}
```

If several applications with the same creator code exist, it is possible to choose which one to register as an implementation of a service by giving a path to the application:

```
xserv::register viewURL Explorer -sig MSIE \
 -path {/Applications/Internet Explorer} -driver {
  AEBuild $params(xservTarget) WWW! OURL \
   ---- [tclAE::build::TEXT $params(url)]
}
```

When a path is given, the signature can be omitted because it is not needed to find the application. However, if you omit the signature, you should set the invocation mode to `App` because it will no longer default to this value. Moreover, if the signature is given, `xserv` will be able to find the application if it is not in the expected folder.

If two or more applications with the same creator code and the same base name exist on your Macintosh, xserv will be able to launch the right one if you gave a path when you registered the application. However, if one of these applications is already running, xserv won't be able to distinguish it from the one you want to use. This is because the `processes` *Alpha* command gives only the base name of the running applications, not their full path.

`xserv::chooseImplementationFor` (proc)

To choose an implementation among the declared ones for a service, we use :

`xserv::chooseImplementationFor` {*xserv name*} {*impl name*} {*group*}?

where {*xserv name*} is the name of the service, and {*impl name*} is the name of the implementation, as given to `xserv::register`. The optional {*group*} allows to choose different implementations of a service in different contexts. For instance, you could define a `View HTML Help` and a `Edit HTML` groups, and choose the Apple Help Viewer for the `View HTML Help` group and Internet Explorer for the `Edit HTML` group. When you

invoke a service for a group, the settings for this group are used. If you don't specify a group, the default group (whose name is the empty string) is used.

For instance, to choose Internet Explorer as the implementation of `viewURL` for the default group, write:

```
xserv::chooseImplementationFor viewURL Explorer
```

To choose Apple Help Viewer as the implementation of `viewURL` for the `View HTML\ Help` group, write:

```
xserv::chooseImplementationFor viewURL Explorer {View HTML Help}
```

When an implementation is chosen for a service, it is validated by `xserv`. The validation ensures that everything needed by the implementation is available.

`xserv::invoke` (proc) To request a service from an XSERV, use :

`xserv::invoke` {*interaction*} {*xserv name*} {*args*}

{*interaction*} tells whether we want the user to be able to interact with the application or not. `-foreground` indicates a possible interaction, and the application will be brought to front. `-background` denies the possibility to interact with the application, which will be launched in the background;

{*xserv name*} is the name of the service to invoke;

{*args*} is a list of key-value pairs, the keys being the parameter names (with an optional leading '-' to make it clear that they are parameter names), and the values being the values of the corresponding parameters for this invocation.

Thus, to display the `http://wwwsi.supelec.fr/` URL with the current implementation of `viewURL`, write:

```
xserv::invoke -foreground viewURL -url http://wwwsi.supelec.fr/
```

`xserv::invokeForGroup` (proc) To request a service from an XSERV in the context of a particular group, use :

`xserv::invokeForGroup` {*group*} {*interaction*} {*xserv name*} {*args*}

which does the same as `xserv::invoke` but uses the current settings for the {*group*} instead of the settings for the default group.

If no implementation has been chosen when a service is invoked, a dialog allows the user to choose an implementation among the registered implementations of the service. If an implementation has been chosen for the default group, it will be pre-selected in the list.

2.2 Invocation hooks

From version 1.2, xserv can call procedures after a service has been invoked. These procedures receive four arguments:

1. the name of the service;
2. the result of the invocation;
3. a list which describes the implementation of the service used for this invocation;
4. the list of paramaters of the invocation.

The last two parameters are key-value lists suitable for use with `array set`.

`xserv::addEndExecHook` (proc)

To add a procedure to the list of procedures which will be called after a service has been invoked, use :

`xserv::addEndExecHook` {*xserv name*} {*proc*}

{*xserv name*} is the name of the service, and {*proc*} is the name of the procedure to call. This procedure must take four parameters as described above.

`xserv::removeEndExecHook` (proc)

To remove a procedure from the list of procedures which will be called after a service has been invoked, use :

`xserv::removeEndExecHook` {*xserv name*} {*proc*}$^{?}$

{*xserv name*} is the name of the service, and {*proc*} is the name of the procedure to remove. If {*proc*} is omitted, all the procedures in the list will be removed.

If the list is empty or does not contain {*proc*}, `removeEndExecHook` will do nothing.

2.3 Categories

From version 1.3, xserv can group services into categories. The only purpose of categories is to help the user navigate the list of services. A service may belong to several categories if this helps the user find the service. For instance, the `dvips` service which is used to produce a PostScript file from a DVI file may belong to the `DVI` and `PS` categories.

`xserv::addToCategory` (proc)

To add services to a category, use :

`xserv::addToCategory` {*category name*} {*xserv name*}$^{+}$

{*category name*} is the name of the category (which will be created if it doesn't exist), and {*xserv name*}$^{+}$are the names of the services to add to the category. A service may be added several times to the same category.

`xserv::removeFromCategory` (proc)

To remove services from a category, use :

`xserv::removeFromCategory` {*category name*} {*xserv name*}$^{+}$

{*category name*} is the name of the category, and {*xserv name*}$^+$are the names of the services to remove from the category. It is not an error to remove a service from a category it doesn't belong to.

`xserv::getCategoriesOf` (proc)

To get the categories a service belongs to, use :

`xserv::getCategoriesOf` {*xserv name*}

{*xserv name*} is the name of a service. The result is the list of the names of all categories the service belongs to. If the service does not belong to a category, the list is empty.

Categories are used to group services when the user wants to choose an implementation for a service using the Set external helpers... item in the Global setup submenu of the Config menu. First, the list of all categories is displayed (including the special category `*\ no category *` which contains all services that do not belong to a category). When the user chooses a category, the list of all services in this category is displayed. Then, when is selects a service, the list of all registered implementations of this service is displayed. If the service has only one mandatory argument, the special item labeled `* Other *` allows to build a generic implementation of the service (see 2.5 on page 8).

If there is only one category, the first dialog is skipped.

2.4 Bundles

From version 1.3, xserv can group services into bundles. A bundle is a set of services that are always provided by the same application, so chosing an implementation for one of the services of the bundle selects the same implementation for all other services in the bundle. When a service is part of a bundle, it is hidden to the end-user who may only choose an implementation for the whole bundle.

`xserv::declareBundle` (proc)

To declare a bundle, use :

`xserv::declareBundle` {*bundle name*} {*desc*} {*xserv*}$^+$

{*bundle name*} is the name of the bundle. This name will appear as a service name to the end-user. {*desc*} is the textual description of the bundle, and {*xserv*}$^+$are the services which are part of the bundle.

Bundles are intended to group the services provided by applications which have a notion of 'session'. Opening the session, performing some operations and closing the session must obviously be performed with the same application, so it doesn't make sense to select different implementations for each of those services. Grouping the services into a bundle hides them to the end-user and makes them appear as a single bundled service.

2.5 Generic implementations

Since version 1.3, xserv allows the creation of generic implementations for services which have only one mandatory argument.

Implementing such services generally amounts to sending an Apple Event to an application, with the argument as direct object, or to execute a command-line containing the

name of the program and the value of the argument. So, if the end-user wants to use an application which is not registered as an implementation of such a single-argument service, xserv can build a generic driver for this application quite easily.

Generic implementations of a service can be created from any dialog which allows to choose an implementation for a single-argument service. In the list of registered implementations of the service, a special item named `* Other *` appears if the service has only one mandatory argument. When this item is chosen, a two-page dialog appears: the first page is for building a generic driver for Apple Event applications, the other page is for building a generic driver for command-line programs.

2.5.1 Building a generic Apple Event driver

For a generic Apple Event driver, the user must enter the name of the application in the "Application" text field. It is also possible to give the signature or creator code of the application between single quotes. For instance, on a Macintosh, using `'ttxt'` will select the TextEdit or the SimpleText application, depending on the version of Mac OS. Using `TextEdit` will select the TextEdit application, and the user will be asked to locate it on the disk.

Then, the user must choose the class and the code of the Apple Event that will be sent to the application. These fields default to `aevt` and `odoc` since this is the most frequent choice. `odoc` may be replaced by `pdoc` if the service is for printing a document.

Last, the type used for the parameter in the Apple Event must be chosen in a pop-up menu between `file` and `text`. The default is `file` since it is the required type for an `odoc` event.

2.5.2 Building a generic command-line driver

For a generic command-line driver, the user must enter the name of the program in the "Program field". This can be the short name of the program or its full path name.

Then, the user must choose the invocation mode among `InSh`, the interactive shell mode, `Shell`, the non-interactive shell mode, and `Exec`, the raw subprocess execution mode. The default is `InSh` since it is the most user-friendly mode for command line tools.

Last, the user must give the general form of the command line for this program in the "Command line" field. The default value is:

`<prog> $params(`{*name*}`)`

`<prog>` will be replaced by the full path to the program when the service is invoked. {*name*} is the name of the only mandatory argument of the service, so `$params(`{*name*}`)` is the value of this argument.

This field can be edited to make the command line suit the syntax required by the program or to add options. For instance, if the argument must be preceded by `-input=` and if we want to use the `-verbose` option of the program, we can write:

`<prog> -verbose -input=$params(`{*name*}`)`

2.5.3 Getting rid of generic implementations

Generic implementations are stored with xserv settings so that they are not lost when *Alpha* quits. However, these implementations are fragile because they are created assuming that an application will understand a given Apple Event, or that a program will understand a particular form of command line.

Since such implementations are created by the end-user, they may be deleted by the end-user in case of an error or if they are no longer needed. The Delete generic implementation... item in the Global setup submenu of the Config menu displays the list of all categories that contain services which have generic implementations. Chosing a category in this dialog displays the list of services that have generic implementations in this category (the first dialog is not displayed if they is only one category). Chosing a service in this dialog displays all generic implementations of this service. Selecting an implementation in this list deletes it after a confirmation dialog has been displayed.

If there are no generic implementations, an alert informs the user, and no other dialog is displayed.

2.6 More about XSERVs

The xserv code defines other commands to work with services. It is possible to suppress a service or a bundle with :

`xserv::forget` (proc)

`xserv::forget` {*xserv name*}

which suppresses the service or bundle named {*xserv name*} and all its implementations. Forgetting a service that does not exist causes no harm. Forgetting a bundle does not forget its members but makes them available as individual services.

The declaration of a service works only if the service is not already declared. This is necessary to avoid that former implementations be invoked and try to use parameters that do not exist any longer in the new service. So, before declaring a service, you can safely use `xserv::forget` to make sure your declaration will be successful.

`xserv::getCurrentImplementationsFor` (proc)

To know which application currently implements a service, use :

`xserv::getCurrentImplementationsFor` {*xserv name*}

which returns a list of {*group*}-{*implementation*} pairs. This list can be used with `array\ set` to set an array containing the current implementation of the service for each group. Each value in this array is itself a list which describes the implementation. This list is suitable for use with `array set` and contains the name of the implementation (as given to `xserv::register`) under the `-name` key. From version 1.3, xserv uses other keys such as `-path` to store the path to the program or application, and `-progs` to store the absolute path to any command-line program needed by the implementation.

Describing the implementation choice with a list allows for future extensions like setting default values for parameters or for the interaction mode when chosing an implementation.

`xserv::listOfServices` (proc)

You can get the list of all known services with :

`xserv::listOfServices` {*which*}?

which returns an alphabeticaly sorted list of the declared services.

{*which*} may take one of the following values:

all asks for the list of all services, including bundles and bundle parts;

bundles is the default value and asks for the list of all services, excluding bundle parts (this is what should be used when presenting the list of services to the end-user);

nobundle asks for the list of all services, excluding bundles (but including their parts). This is the list of all "real" services.

`xserv::getImplementationsOf` (proc)

To get the list of the registered implementations of a service, use :

`xserv::getImplementationsOf` {*xserv name*}

which returns an alphabeticaly sorted list of all registered implementations of {*xserv name*}.

`xserv::describe` (proc)

From the name of a service, you can get its description with :

`xserv::describe` {*xserv name*}

which returns a key-value list describing the service named {*xserv name*}. The possible keys in this list are:

desc the description of the service, as given to `xserv::declare`;

args the argument list of the service, as given to `xserv::declare`;

implementations the list of the registered implementations of this service. This is a key-value list, and each key is the name of an implementation of the service as given to `xserv::register`, while the associated value is a list which describes the implementation (it is the key-value list given to `xserv::register`).

3 The implementation

xserv is an extension for AlphaTcl. It inserts two items in the Global setup submenu of the Config menu which allows to choose the external applications used to implement the declared services, and to suppress generic implementations.

The `quitHook` is used to save the service declarations, implementation registrations and per group implementation choices in the `xservdef.tcl` file in *Alpha*'s preference folder.

xserv adds the search paths set for the "Exec search path" in the miscellaneous package preferences, to the `env(PATH)` global variable so that `exec` finds executables along these paths. the "Exec search path" can also be set in the dialog displayed by the Helper applications... item of the Global setup submenu of the Config menu.

Last, xserv reads its preferences which were stored in the `xservdefs.tcl` file the last time *Alpha* has quit.

Since the `xserv.tcl` Tcl file is extracted from the documentation by the `docstrip` TEX utility, characters which are not pure 7-bit ASCII must be encoded in hexadecimal. Moreover, AlphaTcl is used on several platforms which don't use the same character encodings. So non-ascii character have been encoded in Unicode. `\u2026` is the code for the ellipsis. `\u009d` is the code for "e acute".

```
⟨*tcl⟩

alpha::extension xserv 1.3 {
  menu::insert "globalSetup"  items "helperApplications\u2026" \
            "setExternalHelpers\u2026"  "deleteGenericImplementation\u2026"
  hook::register quitHook ::xserv::saveToPrefs
  ::xserv::fixExecSearchPath
  ::xserv::readPrefs
} maintainer {
  "Fr\u00e9d\u00e9ric Boulanger"  <Frederic.Boulanger@supelec.fr> \
                                    <http://wwwsi.supelec.fr/fb/>  } help {
  xserv defines a new interface to services provided by external\
                                                                  applications.
}

namespace eval xserv {}

```

`xserv::nameFromAppl` (proc)

The `xserv::nameFromAppl` procedure fixes a problem with `nameFromAppl` on Mac OS X where the `/Volumes` mount point is missing for applications stored on another volume than the startup volume. This fix may no longer be necessary when the problem is fixed in AlphaTcl.

```
# proc ::xserv::nameFromAppl {creator} {
#   set path [file split [nameFromAppl $creator]]
#   if {[catch\
          {glob [file join ${file::separator} Volumes [lindex $path 0]]}]} {
#     set path [lrange $path 1 end]
#   } else {
#     set path [linsert $path 0 "Volumes"]
#   }
#   return [eval file join ${file::separator}$path]
# }

```

`xserv::fixExecSearchPath` (proc)

The `xserv::fixExecSearchPath` proc adds the paths in the global variable `execSearchPath` to the `env(PATH)` variable which is used by `exec` to look for executables. The value of the `execSearchPath` variable can be set in the "Miscellaneous packages" package preference dialog. It is set to some arbitrary value in `alphaDefinitions.tcl`, but the `updateExecSearchPath` procedure seems not to be called when the pref is changed.

```
proc ::xserv::fixExecSearchPath {} {
  global execSearchPath env
  if {[info exists execSearchPath]} {
```

```
34      set path [split $env(PATH) ";:"]
35      foreach p $execSearchPath {
36        if {[lsearch -exact $path $p] < 0} {
37          lappend path $p
38        }
39      }
40      set env(PATH) [join $path ":"]
41    }
42 }
43
```

3.1 Managing categories

Categories are used to group services so that it is easier for the end-user to navigate the list of services.

`xserv::addToCategory` (proc)

The proc `xserv::addToCategory` adds services to a category. The category is created if it doesn't exists. The services added to the category may not be declared yet (only their names are used).

```
44 proc ::xserv::addToCategory {cat args} {
```

{*cat*} is the name of the category. All remaining arguments are considered as the names of the services that should be added to the category. A service may be added several times to a category, it will appear only once in this category.

`xserv::categories` ({*category*})

The services in the different categories are stored in the `::xserv::categories` global array which is indexed by the name of the category.

```
45   global ::xserv::categories
46
47   if {![info exists ::xserv::categories($cat)]} {
48     set ::xserv::categories($cat) [list]
49   }
```

Services are added to a category only if they do not already belong to it.

```
50   foreach x $args {
51     if {[lsearch -exact [set ::xserv::categories($cat)] $x] == -1} {
52       lappend ::xserv::categories($cat) $x
53     }
54   }
55 }
```

`xserv::removeFromCategory` (proc)

The proc `xserv::removeFromCategory` removes services from a category.

```
56 proc ::xserv::removeFromCategory {cat args} {
```

{*cat*} is the name of the category. All remaining arguments are considered as the names of the services that should be removed from the category. A service may be removed from a category even if it does not belong to it.

```
57  global ::xserv::categories
58
59  if {[info exists ::xserv::categories($cat)]} {
60    foreach x $args {
61      set idx [lsearch -exact [set ::xserv::categories($cat)] $x]
62      if {$idx != -1} {
63        set ::xserv::categories($cat)\
                    [ lreplace [set ::xserv::categories($cat)] $idx $idx ]
66      }
67    }
68  }
69 }
70
```

`xserv::getCategoriesOf` (proc)

`xserv::getCategoriesOf` returns the list of all categories to which a service belongs (a service may belong to several categories).

```
71 proc ::xserv::getCategoriesOf {xservname} {
72   global ::xserv::categories
73
74   set result [list]
75   if {![info exists ::xserv::categories]} {
76     return $result
77   }
78   foreach cat [array names ::xserv::categories] {
79     if\
          {[lsearch -exact [set ::xserv::categories($cat)] $xservname] != -1}\
                                                                          {
80       lappend result $cat
81     }
82   }
83   return $result
84 }
85
```

3.2 Declaring and forgetting XSERVs

`xserv::declare` (proc)

The `xserv::declare` procedure declares a new XSERV. Each XSERV has a name, a textual description and a set of formal parameter names.

```
86 proc ::xserv::declare {xservname desc args} {
```

{*xservname*} is the name of the new XSERV, {*desc*} is some text that describes what this XSERV is for, and the remaining arguments ⟨*args*⟩ are the names of the parameters of the XSERV. Each item in ⟨*args*⟩ may be either a single {*parameter name*} or a two item list {{*parameter name*} {*default value*}}.

`xserv::services` ({*xserv name*})

The XSERVs are stored in the `::xserv::services` global array which is indexed by the name of the XSERV.

```
87  global ::xserv::services
88
```

If the XSERV already exists, it cannot be declared again (it must be forgotten first). In this case, `xserv::declare` returns `0` to indicate the failure.

```
89  if {[info exists ::xserv::services($xservname)]} {
90    return 0
91  }
92
```

If the XSERV does not exist, we store its declaration in the `::xserv::services` array. This declaration is a list in a form suitable for use with `array set`.

```
93  set ::xserv::services($xservname) [list desc $desc args $args]
94 }
95
```

`xserv::declareBundle` (proc)

The `xserv::declareBundle` procedure declares a bundle of services. All the services in a bundle use the same implementation and are therefore presented to the end-user as a unique bundled service instead of several apparently unrelated distinct services.

```
96 proc ::xserv::declareBundle {bundleName desc args} {
```

{*bundleName*} is the name of the bundle. It plays the same role as the name of a service. {*desc*} is a textual description of the bundle which may help the user understand what this "bundled" service is for. The remaining arguments are the services which are part of the bundle.

```
97  global ::xserv::services
98
99  if {[info exists ::xserv::services($bundleName)]} {
100   array set serv [set ::xserv::services($bundleName)]
101   if {![info exists serv(bundle)]} {
102     error "Declaration of bundle $bundleName overrides a service"
103   }
104 }
105
106 set ::xserv::services($bundleName) [list desc $desc bundle $args]
107}
108
```

`xserv::forget` (proc)

The `xserv::forget` procedure suppresses a service and all its implementations. It is *not* an error to use it for a non-existent service.

When used on a bundle, `xserv::forget` suppresses only the bundle, not the services that were part of it.

```
109 proc ::xserv::forget {xservname} {
110   global ::xserv::services
```

```
111   global ::xserv::currentImplementations
112
113   # Forget the XSERV declaration
114   if {[info exists ::xserv::services($xservname)]} {
115     unset ::xserv::services($xservname)
116   }
117   # Forget the current choices for the XSERV
118   if {[info exists ::xserv::currentImplementations($xservname)]} {
119     unset ::xserv::currentImplementations($xservname)
120   }
121 }
122
```

3.3 Saving and reading settings

xserv:
:saveXservDeclarations
(proc)

The `xserv::saveXservDeclarations` procedure saves the declarations of the XSERVs to a file so that they can be reloaded later.

```
123 proc ::xserv::saveXservDeclarations {file_handle} {
124   global ::xserv::services
125
126   foreach xserv [::xserv::listOfServices "all"] {
127     unset -nocomplain theXserv
128     array set theXserv [set ::xserv::services($xserv)]
129     if {[info exists theXserv(bundle)]} {
130       set decl [list "::xserv::declareBundle" $xserv]
131       lappend decl $theXserv(desc)
132       eval lappend decl $theXserv(bundle)
133     } else {
134       set decl [list "::xserv::declare" $xserv]
135       lappend decl $theXserv(desc)
136       eval lappend decl $theXserv(args)
137     }
138     puts $file_handle $decl
139   }
140 }
141
```

xserv:
:saveXservCategories
(proc)

The `xserv::saveXservCategories` procedure saves the definitions of the categories of services to a file so that they can be reloaded later.

```
142 proc ::xserv::saveXservCategories {file_handle} {
143   global ::xserv::categories
144
145   if {[info exists ::xserv::categories]} {
146     foreach cat [array names ::xserv::categories] {
147       foreach x [set ::xserv::categories($cat)] {
148         puts $file_handle "::xserv::addToCategory $cat $x"
149       }
150     }
151   }
152 }
153
```

xserv:
:saveXservImplementations (proc)

The xserv::saveXservImplementations procedure saves the declarations of the XSERV implementations to a file so that they can be reloaded later.

```
154 proc ::xserv::saveXservImplementations {file_handle} {
155   global ::xserv::services
156
157   foreach xserv [::xserv::listOfServices "nobundle"] {
158     unset -nocomplain theXserv
159     array set theXserv [set ::xserv::services($xserv)]
160     if {[info exists theXserv(implementations)]} {
161       unset -nocomplain theImpls
162       array set theImpls $theXserv(implementations)
163       foreach impl [array names theImpls] {
164         set decl [list "::xserv::register" $xserv $impl]
165         append decl " $theImpls($impl)"
166         puts $file_handle $decl
167       }
168     }
169   }
170 }
171
172
```

xserv:
:saveXservSettings (proc)

The xserv::saveXservSettings procedure saves the group choices of the currently selected implementation for XSERVs, so that these settings can be restored later.

```
173 proc ::xserv::saveXservSettings {file_handle} {
174   global ::xserv::currentImplementations
175
176   foreach xserv [::xserv::listOfServices "nobundle"] {
177     if {[info exists ::xserv::currentImplementations($xserv)]} {
178       unset -nocomplain current
179       array set current [set ::xserv::currentImplementations($xserv)]
180       foreach group [array names current] {
181         set choice [list "::xserv::chooseImplementationFor" $xserv]
182         lappend choice $current($group)
183         lappend choice $group
184         puts $file_handle $choice
185       }
186     }
187   }
188 }
189
```

xserv::saveAll (proc)

The xserv::saveAll procedure saves the whole state of the XSERV package: XSERV declarations, categories, implementation declarations and chosen implementations (per group) for each XSERV.

```
190 proc ::xserv::saveAll {file_handle} {
191   puts $file_handle "# Service declarations"
192   ::xserv::saveXservDeclarations $file_handle
193   puts $file_handle ""
194   puts $file_handle "# Service categories"
195   ::xserv::saveXservCategories $file_handle
```

```
196   puts $file_handle ""
197   puts $file_handle "# Service implementations"
198   ::xserv::saveXservImplementations $file_handle
199   puts $file_handle ""
200   puts $file_handle "# Xserv settings"
201   ::xserv::saveXservSettings $file_handle
202 }
203
```

xserv::saveToPrefs (proc)

The `xserv::saveToPrefs` procedure saves the whole state of the XSERV package into the `xservdefs.tcl` file in the preference folder. A backup of the previous file is saved to `xservdefs.bak`.

```
204 proc ::xserv::saveToPrefs {} {
205   global PREFS
206
207   if {[file exists [file join $PREFS xservdefs.tcl]]} {
208     file rename -force [file join $PREFS xservdefs.tcl] \
                                              [file join $PREFS xservdefs.bak]
210   }
211
212   set err [catch {open "[file join $PREFS xservdefs.tcl]" "w"} pref_file]
213   if {$err != 0} {
214     alertnote "Could not save your helper application settings! ($err)"
215     return
216   }
217
218   ::xserv::saveAll $pref_file
219   close $pref_file
220 }
221
```

xserv::readPrefs (proc)

The `xserv::readPrefs` procedure restores the state of the XSERV package to the state saved in the `xservdefs.tcl` file in the preference folder.

```
222 proc ::xserv::readPrefs {} {
223   global PREFS
224
225   if {[file exists "[file join $PREFS xservdefs.tcl]"]} {
226     source "[file join $PREFS xservdefs.tcl]"
227   }
228 }
229
```

3.4 Getting information about the XSERVs

xserv::listOfServices (proc)

The `xserv::listOfServices` procedure returns the list of all declared XSERVs. It builds this list from the names in the `::xserv::services` array. The list is sorted so that it can be used to let the user pick an XSERV in a dialog.

```
230 proc ::xserv::listOfServices { {which "bundles"} } {
```

{*which*} tells which services we want in the list. It can take one of the following values:

all asks for all services, including bundles and their parts;

bundles asks for all services, excluding bundle parts (this the list an end-user should see);

nobundle asks for all services, excluding bundles (this is the list of all “real” services).

```
231  global ::xserv::services
232  set servs [lsort -dictionary [array names ::xserv::services]]
233  if {$which == "all"} {
234    return $servs
235  } elseif {$which == "bundles"} {
236    foreach serv [array names ::xserv::services] {
237      array set s [set ::xserv::services($serv)]
238      if {[info exists s(bundle)]} {
239        foreach sub $s(bundle) {
240          set idx [lsearch -exact $servs $sub]
241          if {$idx != -1} {
242            set servs [lreplace $servs $idx $idx]
243          }
244        }
245      }
246    }
247    return $servs
248  } elseif {$which == "nobundle"} {
249    foreach serv [array names ::xserv::services] {
250      unset -nocomplain s
251      array set s [set ::xserv::services($serv)]
252      if {[info exists s(bundle)]} {
253        set idx [lsearch -exact $servs $serv]
254        if {$idx != -1} {
255          set servs [lreplace $servs $idx $idx]
256        }
257      }
258    }
259    return $servs
260  } else {
261    error "Unknown value $which for argument of ::xserv::listOfServices"
262  }
263 }
264
```

`xserv::describe` (proc) The `xserv::describe` procedure returns the description of an XSERV in the form of an empty list if the XSERV does not exist or a list suitable for use with `array set` containing the following entries:

1 `desc`: the textual description of the XSERV;

2 `params`: the list of the parameters of the XSERV, with default value when applicable (the parameter is then a two element list, the first element being the name of the parameter and second its default value);

3 `implementations`: the list of all registered implementations of the service. This is a key-value list, and the entries are the registered names of the implementations. The associated values are `array set`-like lists with as many entries as necessary to describe the corresponding implementation. Usual entries are:

- `-mode`: gives the invocation mode ({*App*} for a MacOS application, {*Exec*} for a Unix application, {*Shell*} for a command line to be interpreted by a shell, {*Alpha*} for a Tcl script to be evaluated by *Alpha*);

1 `-sig`: the signature of the application, if applicable;

2 `-path`: the path to the application, if applicable;

3 `-driver`: the script that drives the application according to the parameters of the XSERV.

```
proc ::xserv::describe {xservname} {
  global ::xserv::services

```

The `::xserv::services` global array contains the declarations of the XSERVs, as seen in 3.2.

The description is an empty list if the XSERV does not exists:

```
  if {![info exists ::xserv::services($xservname)]} {
    return [list]
  }

```

If the XSERV exists, we just return its description:

```
  return [set ::xserv::services($xservname)]
}

```

`xserv::getBundleName` (proc)

`xserv::getBundleName` returns the name of the bundle thah contains a service, or an empty string if the service is not part of a bundle.

```
proc ::xserv::getBundleName {xservname} {
  global ::xserv::services

  foreach s [array names ::xserv::services] {
    unset -nocomplain serv
    array set serv [set ::xserv::services($s)]
    if {![info exists serv(bundle)]} {
      continue
    }
    if {[lsearch -exact $serv(bundle) $xservname] != -1} {
      return $s
    }
  }
  return ""
}

```

`xserv::isBundle` (proc)

`xserv::isBundle` tells if a service is a bundle or a “real” service.

```
291 proc ::xserv::isBundle {xservname} {
292   global ::xserv::services
293
294   if {![info exists ::xserv::services($xservname)]} {
295     error "Undeclared service \"$xservname\""
296   }
297   array set serv [set ::xserv::services($xservname)]
298   return [info exists serv(bundle)]
299 }
300
```

`xserv:`
`:getImplementationsOf`
(proc)

The `xserv::getImplementationsOf` procedure returns the list of the names of all registered implementations of the XSERV `xservname`. This list is sorted so that it can be used to let the user pick an implementation in a dialog.

```
301 proc ::xserv::getImplementationsOf {xservname} {
302   global ::xserv::services
303
304   if {![info exists ::xserv::services($xservname)]} {
305     error "Undeclared service \"$xservname\""
306   }
307   array set theXserv [set ::xserv::services($xservname)]
308   if {[info exists theXserv(bundle)]} {
309     foreach serv $theXserv(bundle) {
310       set impls($serv) [::xserv::getImplementationsOf $serv]
311     }
312     set first [lindex [array names impls] 0]
313     set servs [lrange [array names impls] 1 end]
314     set implementations [list]
315     foreach imp $impls($first) {
316       set common 1
317       foreach serv $servs {
318         if {[lsearch -exact $impls($serv) $imp] == -1} {
319           set common 0
320           break
321         }
322       }
323       if {$common} {
324         lappend implementations $imp
325       }
326     }
327     return [lsort -dictionary $implementations]
328   }
329
330   if {[info exists theXserv(implementations)]} {
331     array set implementations $theXserv(implementations)
332   } else {
333     array set implementations [list]
334   }
335   return [lsort -dictionary [array names implementations]]
336 }
337
```

xserv::getCurrentImplementationsFor (proc)

The `xserv::getCurrentImplementationsFor` procedure returns the name of the implementations that are used as current implementations of an XSERV. The result is a key-value list (suitable for use with `array set`) with the groups as keys and the current implementation for the group as value. The current implementation for a group is a key-value list. The only mandatory key for now is `-name` which identifies the name of the implementation, as given to `xserv::register`. Other keys may be used to extend the notion of "current implementation".

As of version 1.3, the `-path` key identifies the absolute path of the application or program used for the implementation, and the `-progs` key identifies the list of the absolute paths of the command-line programs needed by the implementation.

```
proc ::xserv::getCurrentImplementationsFor {xservname} {
  global ::xserv::services
  global ::xserv::currentImplementations

  if {![info exists ::xserv::services($xservname)]} {
    error "Undeclared service \"$xservname\""
  }
```

It is an error to call this procedure on an XSERV that does not exist. If the XSERV exists and no application has been chosen for it yet, we return an empty list:

```
  if {[info exists ::xserv::currentImplementations($xservname)]} {
    return [set ::xserv::currentImplementations($xservname)]
  } else {
    return [list]
  }
}

```

3.5 Working with implementations

xserv::register (proc)

The `xserv::register` procedure registers an implementation of an xserv.

{*xservname*} is the name of the implemented XSERV.

{*implName*} is the name we want to use to refer to the implementing application.

⟨*args*⟩ is a `array get`-like list which describes the implementation.

For instance, if application CMacTeX of signature "*XeT" supports the "tex" XSERV, it can be registered with the following call:

```
::xserv::register tex CMacTeX -sig *XeT -driver {
  buildNewCMacTeXAE "tex $options &$format" $filename
}
```

The {*-sig*} argument indicates that this implementation is identified by a MacOS creator code (`*TeX` here). The {*-driver*} argument says that to implement the "tex" XSERV with CMacTeX, one should execute `buildNewCMacTeXAE ...`. The driver script can get the

values of the arguments to the service invocation in the `params` array. `params` is the only special name introduced by xserv in the context where the script is executed.

Two additional arguments are always added to the `params` array :

- `xservTarget` contains the value of the Apple Event target when the implementation is a Mac OS application (`App` invocation mode);
- `xservInteraction` indicates whether the implementation should be put to the foreground (1) or let in the background (0).

To allow future extensions of xserv, all parameter names beginning with `xserv` are reserved.

From version 1.3, when an implementation is registered with a list of programs (using the `-prog` key), the absolute path to each program {*prog*} of the list is available in the `xserv-`{*prog*} entry of the `params` array.

```
352 proc ::xserv::register {xservname implName args} {
353   global ::xserv::services
354
355   if {![info exists ::xserv::services($xservname)]} {
356     error "Undeclared service \"$xservname\""
357   }
358
359   array set theXserv [set ::xserv::services($xservname)]
360   if {[info exists theXserv(bundle)]} {
361     error "Attempt to register an implementation for bundle $xservname"
362   }
363   if {[info exists theXserv(implementations)]} {
364     array set theImpls $theXserv(implementations)
365   } else {
366     array set theImpls [list]
367   }
368
369   # Remember the description of the implementation.
370   if {[llength $args] % 2 != 0} {
371     error {description must be "[key value]*"}
372   }
373   array set impDesc $args
374   if {[llength [array names impDesc]] !=\
                                        [llength [array names impDesc -*]]} {
375     error {Vince said that keys must have a leading '-'}
376   }
377
378   if {![info exists impDesc(-mode)]} {
379     if {[info exists impDesc(-sig)]} {
380       set impDesc(-mode) App
381     } elseif {[info exists impDesc(-shell)]} {
382       set impDesc(-mode) Shell
383     } else {
384       set impDesc(-mode) Exec
385     }
386   }
```

```
387
388   set theImpls($implName) [array get impDesc]
389   set theXserv(implementations) [array get theImpls]
390   set ::xserv::services($xservname) [array get theXserv]
391 }
392
```

xserv:
:forgetImplementation
(proc)

The `xserv::forgetImplementation` procedure unregisters an implementation of an xserv. This procedure cannot be used with bundles since the implementations of a bundled service are "virtual" (they are the implementations which are common to all the members of the bundle).

```
393 proc ::xserv::forgetImplementation {xservname implName} {
394   global ::xserv::services
395   global ::xserv::currentImplementations
396
397   if {![info exists ::xserv::services($xservname)]} {
398     error "Undeclared service \"$xservname\""
399   }
400   array set theXserv [set ::xserv::services($xservname)]
401   if {[info exists theXserv(bundle)]} {
402     error "Attempt to forget an implementation of bundle $xservname"
403   }
404   if {![info exists theXserv(implementations)]} {
405     array set theImpls [list]
406   } else {
407     array set theImpls $theXserv(implementations)
408   }
409   if {![info exists theImpls($implName)]} {
410     error "Unknown implementation \"$implName\" for service\
                                                          \"$xservname\""
411   }
412   unset theImpls($implName)
413   set theXserv(implementations) [array get theImpls]
414   set ::xserv::services($xservname) [array get theXserv]
415
416   if {![info exists ::xserv::currentImplementations($xservname)]} {
417     return
418   }
419   array set current [set ::xserv::currentImplementations($xservname)]
420   foreach group [array names current] {
421     unset -nocomplain imp
422     array set imp $current($group)
423     if {"$imp(-name)" == "$implName"} {
424       unset current($group)
425     }
426   }
427   set ::xserv::currentImplementations($xservname) [array get current]
428 }
429
```

xserv:
:chooseImplementationFor
(proc)

The `xserv::chooseImplementationFor` procedure allows to choose the implementation to use for an XSERV, among the registered implementations. Several settings may

be remembered for different groups of users, or clients of the service. A default group is used when no group is specified.

For instance: `::xserv::chooseImplementationFor tex CMacTeX` chooses CMacTeX to implement the `tex` service for the default group, while `::xserv::chooseImplementationFor\ tex teTeX docgen` chooses teTeX to implement the `tex` service for the `docgen` group. This implementation will be used when the `tex` service is invoked for the `docgen` group, while CMacTeX will be used when no particular group is specified.

The `implName` argument may be a single item, in which case it is considered as the name of the chosen implementation. It may also be a key-value list if data other than the name of the implementation must be associated to the choice. This key-value list must contain a `-name` key with the name of the implementation as its value.

The special implementation name `* Other *` is used to build generic implementations for services which have only one mandatory argument.

For instance `::xserv::chooseImplementationFor tex {-name CMacTeX -format\ hlatex}` may be used to choose CMacTeX as the implementation of the `tex` service and to give `hlatex` as the default format to use.

In the current version of xserv, only the `-name`, `-path` and `-progs` keys may be interpreted, but the list structure of the `implName` argument allows for future extensions.

```
proc ::xserv::chooseImplementationFor {xservname implName {group ""}} {
  global ::xserv::services
  global ::xserv::currentImplementations

  if {![info exists ::xserv::services($xservname)]} {
    error "Undeclared service \"$xservname\""
  }

```

If the name of the implementation is `* Other *`, build a generic implementation and return it.

```
  if\
      {([llength $implName] == 1) && ([lindex $implName 0] == "* Other *")}\
                                                                        {
    set implName [::xserv::addGenericImplementation $xservname]
    if {[llength $implName] == 0} {
      return $implName
    }
  }

```

If the service is a bundle, we must set the implementations of all the members of the bundle.

```
  array set theXserv [set ::xserv::services($xservname)]
  if {[info exists theXserv(bundle)]} {
    return [::xserv::chooseImplementationForBundle  $xservname $implName\
                                                                  $group]
  }
  if {[info exists theXserv(implementations)]} {
```

```
451     array set theImpls $theXserv(implementations)
452   } else {
453     array set theImpls [list]
454   }
455
456   if {[llength $implName] == 1} {
457     set implName [list -name [lindex $implName 0]]
458   } elseif {[llength $implName] % 2 != 0} {
459     error {Malformed parameter list. Must be [key value]*}
460   }
461   array set implChoice $implName
462   if {[llength [array names implChoice]]  !=\
                                    [llength [array names implChoice -*]]} {
464     error {Vince said that keys must have a leading '-'}
465   }
466   if {![info exists implChoice(-name)]} {
467     error "No -name key in [array get implChoice]"
468   }
469   set implName $implChoice(-name)
470   if {![info exists theImpls($implName)]} {
471     error "No implementation named $implName for service \"$xservname\""
472   }
473
474   array set chosenImpl $theImpls($implName)
475   set chosenMode $chosenImpl(-mode)
476
477   if {[info commands ::xserv::validateImpChoice$chosenMode] != ""} {
478     set validated [  ::xserv::validateImpChoice$chosenMode \
                              [array get implChoice]  $theImpls($implName) ]
483     if {[llength $validated] == 0} {
484       return $validated
485     } else {
486       unset implChoice
487       array set implChoice $validated
488     }
489   }
490
```

When the implementation of a service is changed, we call the global implementation-change hooks and the implementation-change hooks which are specific of this service.

```
491   hook::callAll xserv::implChangeHook "" $xservname $group \
                                                      [array get implChoice]
493   hook::callAll xserv::implChangeFor${xservname}Hook "" $xservname\
                                            $group  [array get implChoice]
495
496   if {[info exists ::xserv::currentImplementations($xservname)]} {
497     array set current [set ::xserv::currentImplementations($xservname)]
498   } else {
499     array set current [list]
500   }
501   set current($group) [array get implChoice]
502   set ::xserv::currentImplementations($xservname) [array get current]
503   return [array get implChoice]
```

```
}
```

xserv::chooseImplementationForBundle (proc)

The `xserv::chooseImplementationForBundle` procedure is used to choose an implementation for a bundle of services. This amounts to choose the same implementation for each member of the bundle.

```
proc ::xserv::chooseImplementationForBundle\
                                        {bundleName implName {group ""}} {
  global ::xserv::services

  if {![info exists ::xserv::services($bundleName)]} {
    error "Undeclared bundle \"$bundleName\""
  }
  array set theXserv [set ::xserv::services($bundleName)]
  if {![info exists theXserv(bundle)]} {
    error "$bundleName is not a bundle"
  }
  foreach serv $theXserv(bundle) {
    set implName [::xserv::chooseImplementationFor $serv $implName $group]
    if {[llength $implName] == 0} {
      return $implName
    }
  }
  if {[info exists ::xserv::currentImplementations($bundleName)]} {
    array set current [set ::xserv::currentImplementations($bundleName)]
  } else {
    array set current [list]
  }
  set current($group) [array get implName]
  set ::xserv::currentImplementations($bundleName) [array get current]
  return $implName
}
```

3.6 Generic implementations

Generic implementations allow the end-user to implement a service with an application or a program which is not registered as an implementation of this service. Generic implementations are restricted to services which have only one mandatory argument, so that the driver of the implementation uses a simple 'aevt'/'odoc' apple event or a "prog argument" command line.

xserv::addGenericImplementation (proc)

The `xserv::addGenericImplementation` procedure displays the dialog used to create a generic implementation of a service. This dialog has two pages, one for Apple Event driven applications, the other for command-line programs.

```
proc ::xserv::addGenericImplementation {xservname} {
  set mandatory [::xserv::mandatoryArgsOf $xservname]
```

Generic implementation for services with more than one mandatory argument are too

complex and too error prone, so they are not supported.

```
534  if {[llength $mandatory] != 1} {
535    set msg "Cannot build generic implementation for $xservname: "
536    append msg "too many arguments."
537    error $msg
538  }
539  set mandatory [lindex $mandatory 0]
540  set dstuff [list  {Apple Events}\
          {-path "" -class aevt -event odoc -type "" kind ""}  {Command line}\
         [list  -path "" -mode "" -cmd "<prog> \$params($mandatory)" kind "" \
                                                                          ]  ]
546  set page [lindex $dstuff 0]
547  while {![catch {set dstuff [dialog::make_paged -defaultpage $page \
        [list [lindex $dstuff 0] [lindex $dstuff 1]  {{-path var Application}\
                        {-class var {Event class}}  {-event var {Event code}}\
             {-type {menu {file text}} "parameter type"}  {kind thepage}}  ] \
                                [list [lindex $dstuff 2] [lindex $dstuff 3] \
                     {{-path var Program} {-mode {menu {InSh Shell Exec}} Mode} \
                              {-cmd var "Command line"} {kind thepage}}  ]]}]} {
557    array set answ $dstuff
558    array set newImp $answ(Apple Events)
559    set page $newImp(kind)
560    if {$page == "Command line"} {
561      set imp [::xserv::addGenericCommandLine $xservname $mandatory \
                                                          $answ(Command line)]
563    } else {
564      set imp [::xserv::addGenericAppleEvents $xservname $mandatory \
                                                          $answ(Apple Events)]
566    }
567    if {[llength $imp] > 0} {
568      return $imp
569    }
570  }
571  return [list]
572 }
573
```

xserv::mandatoryArgsOf (proc)

`xserv::mandatoryArgsOf` returns the list of all mandatory arguments of a service (the arguments which don't have a default value).

```
574 proc ::xserv::mandatoryArgsOf {xservname} {
575   global ::xserv::services
576
577   if {![info exists ::xserv::services($xservname)]} {
578     error "Undeclared service \"$xservname\""
579   }
580   array set theXserv [set ::xserv::services($xservname)]
581   set mandatory [list]
582   foreach a $theXserv(args) {
583     if {[llength $a] == 1} {
584       lappend mandatory [lindex $a 0]
585     }
586   }
```

```
587   return $mandatory
588 }
589
```

xserv:
:addGenericCommandLine
(proc)

`xserv::addGenericCommandLine` processes the data from the "command line" page of the generic-implementation dialog to register a generic command-line implementation.

```
590 proc ::xserv::addGenericCommandLine {xservname param impl} {
591   array set theImpl $impl
592   if {$theImpl(-path) == ""} {
593     alertnote "You must give a program name or path"
594     return [list]
595   }
596   if {$theImpl(-cmd) == ""} {
597     alertnote "You must define the command line"
598     return [list]
599   }
600   set progName [file tail $theImpl(-path)]
601   set implName "generic-$progName"
602   set decl\
          [list ::xserv::register $xservname $implName -mode $theImpl(-mode)]
603   if {[lsearch -exact {InSh Shell} $theImpl(-mode)] != -1} {
604     lappend decl -shell sh
605   }
606   set fixedProg 1
607   if {   !([file pathtype $theImpl(-path)] == "absolute")  ||\
                                        ![file executable $theImpl(-path)]} {
609     lappend decl -progs [list $progName]
610     set fixedProg 0
611   }
612   if {$fixedProg} {
613     regsub {<prog>} $theImpl(-cmd) $theImpl(-path) driver
614   } else {
615     regsub {<prog>} $theImpl(-cmd) "\$params(xserv-$progName)" driver
616   }
617   lappend decl -driver "return \"$driver\""
```

Generic implementations are tagged with a `-generic` key so that they van be distinguished from “supported” implementations. The value associated to this key is the registration date of the implementation in ISO format.

```
618   lappend decl -generic [mtime now iso]
619   eval $decl
620   return [list -name $implName]
621 }
622
```

xserv:
:addGenericAppleEvents
(proc)

`xserv::addGenericAppleEvents` processes the data from the "Apple Event" page of the generic-implementation dialog to register a generic Apple Event implementation.

```
623 proc ::xserv::addGenericAppleEvents {xservname param impl} {
624   array set theImpl $impl
625   if {$theImpl(-path) == ""} {
```

```
626     alertnote "You must give an application name or signature"
627     return [list]
628   }
629   if {$theImpl(-class) == ""} {
630     alertnote "You must define the Apple Event class"
631     return [list]
632   }
633   if {$theImpl(-event) == ""} {
634     alertnote "You must define the Apple Event"
635     return [list]
636   }
637   set progName [file tail $theImpl(-path)]
638   set implName "generic-$progName"
639   set decl [list ::xserv::register $xservname $implName -mode App]
640   if {[regexp {^'....'$} $theImpl(-path)]} {
641     lappend decl -sig $theImpl(-path)
642   } else {
643     lappend decl -path $theImpl(-path)
644   }
645   set driver "tclAE::send -p \$params(xservTarget) "
646   append driver "$theImpl(-class) $theImpl(-event) ---- "
647   if {$theImpl(-type) == "file"} {
648     append driver "\[tclAE::build::alis "
649   } else {
650     append driver "\[tclAE::build::TEXT "
651   }
652   append driver "\$params($param)\]"
653   lappend decl -driver $driver
654   lappend decl -generic [mtime now iso]
655   eval $decl
656   return [list -name $implName]
657 }
658
```

xserv:
:getGenericImplementationsOf
(proc)

`xserv::getGenericImplementationsOf` returns the list of all generic implementations of a service. Generic implementations are identified by the presence of a `-generic` key in their definition.

```
659 proc ::xserv::getGenericImplementationsOf {xservname} {
660   global ::xserv::services
661
662   if {![info exists ::xserv::services($xservname)]} {
663     error "Unknown service \"$xservname\""
664   }
665   array set serv [set ::xserv::services($xservname)]
666   if {![info exists serv(implementations)]} {
667     return [list]
668   }
669   set gen [list]
670   array set impls $serv(implementations)
671   foreach imp [array names impls] {
672     unset -nocomplain theImp
673     array set theImp $impls($imp)
674     if {[info exists theImp(-generic)]} {
```

```
675       lappend gen $imp
676     }
677   }
678   return $gen
679 }
680
```

xserv::deleteGenerics (proc)

`xserv::deleteGenerics` displays a dialog to let the user select a generic implementation and delete it.

```
681 proc ::xserv::deleteGenerics {} {
682   global ::xserv::categories
683   set servs [::xserv::listOfServices]
```

From the list of services, we build the list of services which have at least one generic implementation.

```
684   set noCat [list]
685   foreach s $servs {
686     if {[llength [::xserv::getGenericImplementationsOf $s]] > 0} {
687       if {[lsearch -exact $noCat $s] == -1} {
688         lappend noCat $s
689       }
690     }
691   }
```

Then, we sort these implementations according to the category of the service they implement.

```
692   if {[info exists ::xserv::categories]} {
693     foreach cat [array names ::xserv::categories] {
694       set avails [list]
695       foreach serv [set ::xserv::categories($cat)] {
696         if {[lsearch -exact $noCat $serv] != -1} {
697           lappend avails $serv
698         }
699       }
700       if {[llength $avails] > 0} {
701         set editCats($cat) $avails
702       }
703     }
```

We remove the categorized implementations from the `noCat` list of the implementations with no category.

```
704     if {[info exists editCats]} {
705       foreach cat [array names editCats] {
706         foreach serv $editCats($cat) {
707           set idx [lsearch -exact $noCat $serv]
708           if {$idx != -1} {
709             set noCat [lreplace $noCat $idx $idx]
710           }
711         }
712       }
```

If there are implementations with no category, we add a special `* no category *` category for them.

```
713       if {[llength $noCat] > 0} {
714         set editCats(*\ no\ category\ *) $noCat
715       }
716     }
717   }
718
719   if {[info exists editCats]} {
720     set theCats [array names editCats]
721     while 1 {
722       if {[llength $theCats] > 1} {
723         set status [catch {set theCat [  listpick -p "Choose a category\
                                of services"  [lsort -dictionary $theCats]  ]}]
727         if {$status} {
728           return
729         }
730       } else {
731         set theCat [lindex $theCats 0]
732       }
733       set status [catch {set theXSERV [  listpick -p "Choose a service" \
                                    [lsort -dictionary $editCats($theCat)]  ]}]
737       if {$status} {
738         if {[llength $theCats] == 1} {
739           return
740         }
741       } else {
742         break
743       }
744     }
745   } else {
746     if {[llength $noCat] == 0} {
747       alertnote "There are no generic implementations."
748       return
749     }
750     set status [catch {set theXSERV\
           [  listpick -p "Choose a service"  [lsort -dictionary $noCat]  ]}]
754     if {$status} {
755       return
756     }
757   }
758   set generics [::xserv::getGenericImplementationsOf $theXSERV]
759   set status [catch {set theImp [  listpick -p "Choose an implementation\
                                to delete"  [lsort -dictionary $generics]  ]}]
763   if {$status} {
764     return
765   }
```

We ask the user to confirm the deletion of the implementation.

```
766   set question "Delete generic implementation\n"
767   append question "  $theImp\n"
768   append question "of service\n"
```

```
769   append question "  $theXSERV ?"
770   if {[askyesno $question] == "yes"} {
771     ::xserv::forgetImplementation $theXSERV $theImp
772   }
773 }
774
```

xserv:
:deleteGenericImplementation
(proc)

The procedure `deleteGenericImplementation` is a wrapper around `xserv::deleteGenerics`, and is used as the callback for the "Delete generic implementation" menu item.

```
775 proc deleteGenericImplementation {} {
776   ::xserv::deleteGenerics
777 }
778
```

3.7 Validating implementation choices

Since version 1.3 of xserv, an implementation is validated when the user chooses it, and before it is invoked.

The validation process should ensure that everything needed by the implementation is available. Information gathered during validation and which will be used at invocation time should be stored in the implementation choice, which is a key-value list.

A validation procedure is called automatically if it exists. When an implementation is selected or is about to be invoked, xserv looks for a procedure named `xserv::validateImpChoice`{*mode*}, where {*mode*} is the invocation mode of the implementation. If no such procedure exists, the implementation is considered to be valid. If the procdure exists, it is called with two arguments: the implementation choice and the registered implementation.

The procedure should return an empty list if the implementation could not be validated, or a key-value list containing all necessary information for an invocation to succeed.

xserv:
:validateImpChoiceApp
(proc)

The procedure `xserv::validateImpChoiceApp` validates an implementation choice for the `App` invocation mode. It checks that the application exists and adds its absolute path to the implementation choice under the `-path` key.

```
779 proc ::xserv::validateImpChoiceApp {choice impl} {
780   array set theChoice $choice
```

If `choice` has a `-path` key which leads to an existing file with type 'APPL' (or with no type on Mac OS X), we consider it as a valid implementation.

```
781   if {[info exists theChoice(-path)] && [file exists $theChoice(-path)]} {
782     getFileInfo $theChoice(-path) appInfo
783     if {$appInfo(type) == "APPL" || $appInfo(type) == ""} {
784       return $choice
785     }
786   }
```

If `choice` has no `-path` key, we must look into the registered implementation for either a path or a signature. The first piece of information which leads to an existing application is used.

```
787  array set theImpl $impl
788  if {[info exists theImpl(-path)] && [file exists $theImpl(-path)]} {
789    getFileInfo $theImpl(-path) appInfo
790    if {$appInfo(type) == "APPL" || $appInfo(type) == ""} {
791      set theChoice(-path) $theImpl(-path)
792      return [array get theChoice]
793    }
794  }
795  if {[info exists theImpl(-sig)]} {
796    if {![catch {set app [nameFromAppl $theImpl(-sig)]}]} {
797      set theChoice(-path) $app
798      return [array get theChoice]
799    }
800  }
```

If we couldn't find an application (the path is wrong, or the Finder data base couldn't give a path from the signature), ask the user to locate the application. Remove a possible ".app" extension on Mac OS X.

```
801  set prompt "Locate application"
802  if {[info exists theImpl(-path)]} {
803    set appName [file tail $theImpl(-path)]
804    regsub {\.app$} $appName {} appName
805    append prompt " $appName"
806  }
807  if {[info exists theImpl(-sig)]} {
808    set appSig $theImpl(-sig)
809    append prompt " with type $appSig"
810  }
811  if {[catch {set appPath [getfile $prompt]}]} {
812    return [list]
813  } else {
814    set theChoice(-path) $appPath
815    return [array get theChoice]
816  }
817 }
818
```

`xserv::validateImpChoiceExec` (proc)

The procedure `xserv::validateImpChoiceExec` validates an implementation choice for the `Exec` invocation mode. It checks that the program exists and adds its absolute path to the implementation choice under the `-path` key. It also checks that all the programs listed under the `-progs` key exists and adds their absolute paths to the implementation choice under the `-progs` key.

```
819 proc ::xserv::validateImpChoiceExec {choice impl} {
820   array set theImpl $impl
```

For each program name under the `-progs` key in the implementation registration, check

that a valid program is available in the key-value list under the `-progs` key in the implementation choice.

```
821  if {[info exists theImpl(-progs)] && [llength $theImpl(-progs)] != 0} {
822    array set theChoice $choice
823    set nprogs [llength $theImpl(-progs)]
824    if {[info exists theChoice(-progs)]} {
825      array set theProgs $theChoice(-progs)
826    } else {
827      array set theProgs [list]
828    }
829
830    set foundProgs [list]
831    for {set i 0} {$i < $nprogs} {incr i} {
832      set pname [lindex $theImpl(-progs) $i]
833      if {[info exists theProgs($pname)]} {
834        set found [::xserv::validateProg $theProgs($pname) $pname]
835      } else {
836        set found [::xserv::validateProg "" $pname]
837      }
838      if {$found != ""} {
839        lappend foundProgs $pname $found
840      } else {
841        return [list]
842      }
843    }
844    set theChoice(-progs) $foundProgs
845    return [array get theChoice]
846  } else {
847    return $choice
848  }
849 }
850
```

`xserv::validateImpChoiceShell` (proc)

The procedure `xserv::validateImpChoiceShell` validates an implementation choice for the `Shell` invocation mode. After checking that the shell to use exists, it calls `validateImpChoiceExec` to validate the `-progs` aspect.

```
851 proc ::xserv::validateImpChoiceShell {choice impl} {
852   array set theChoice $choice
853   array set theImpl $impl
854   if {[info exists theChoice(-shell)]} {
855     set curChoice $theChoice(-shell)
856   } else {
857     set curChoice ""
858   }
859   set curChoice [::xserv::validateProg $curChoice $theImpl(-shell)]
860   if {$curChoice == ""} {
861     return [list]
862   } else {
863     set theChoice(-shell) $curChoice
864   }
865
866   ::xserv::validateImpChoiceExec [array get theChoice] $impl
```

```
867 }
868
```

`xserv::validateImpChoiceInSh` (proc)

The procedure `xserv::validateImpChoiceInSh` validates an implementation choice for the `InSh` interactive invocation mode. This is just the same as vaidating for `Shell` mode.

```
869 proc ::xserv::validateImpChoiceInSh {choice impl} {
870   ::xserv::validateImpChoiceShell $choice $impl
871 }
872
```

`xserv::validateProg` (proc)

The procedure `xserv::validateProg` validates a program path against a program name. It returns the absolute path to the program or an empty string if the program could not be found.

```
873 proc ::xserv::validateProg {prog name} {
874   global env
875
```

{*prog*} is the path to the program, {*name*} is the program to find.

If {*prog*} is executable, assume it is a good choice for {*name*}.

```
876   if {$prog != "" && [file executable $prog]} {
877     return $prog
878   }
```

Look for programs named {*name*} in the command path. If none are found, ask the user to locate the program. If only one is found, return it. If more than one are found, ask the user to choose among them.

```
879   set candidates [::xserv::findProg $name [split $env(PATH) ";:"]]
880   if {[llength $candidates] == 0} {
881     alertnote "There is no \'$name\'program in your PATH."
882     if {[catch {set prog [getfile "Locate program $name"]}]} {
883       return ""
884     } else {
885       return $prog
886     }
887   } elseif {[llength $candidates] == 1} {
888     return [lindex $candidates 0]
889   } else {
890     if {[catch {set prog\
              [  listpick -p "Choose a program for $name"  $candidates  ]}]} {
894       return ""
895     } else {
896       return $prog
897     }
898   }
899 }
900
```

`xserv::findProg` (proc) The procedure `xserv::findProg` searches a program in a list of directories.

```
901 proc ::xserv::findProg {prog pathlist {exact 0}} {
```

{*prog*} is the program to find. {*pathlist*} is the list of the directories in which to search. {*exact*} tells if {*prog*} is the exact name of the program to find. When {*exact*} is 1, we must just check that {*prog*} is executable. When {*exact*} is 0, we can look for another program with the same tail name.

```
902   if {[file pathtype $prog] == "absolute"} {
903     if {[file executable $prog]} {
904       return [list $prog]
905     } elseif {$exact} {
906       return [list]
907     } else {
908       set prog [file tail $prog]
909     }
910   }
911   set candidates [list]
912   foreach path $pathlist {
913     set p [file join $path $prog]
914     if {[file executable $p] && [lsearch -exact $candidates $p] < 0} {
915       lappend candidates $p
916     }
917   }
918   return $candidates
919 }
920
```

3.8 End user interface

`xserv::selectImplementationFor` (proc) The `xserv::selectImplementationFor` procedure asks the user which implementation of an XSERV he wants to use. This procedure is called when an XSERV is invoked but no implementation has been chosen for it yet, or when the user selects the Set external helpers menu item to choose an implementation for a service.

If an implementation has already been chosen for the XSERV, `xserv::selectImplementationFor` makes it the default selection in the list of implementations.

If the service has only one mandatory argument, a special implementation labelled `*\ Other *` is added to the list of implementations and allows the creation of a generic implementation.

If the service is part of a bundle, the user is asked to chose an implementation for the bundle (end-users should not see services which are part of a bundle).

```
921 proc ::xserv::selectImplementationFor {xservname {group ""}} {
922   global ::xserv::currentImplementations
923   global ::xserv::services
924
925   if {![info exists ::xserv::services($xservname)]} {
926     error "Undeclared service \"$xservname\""
927   }
```

```
928
929   set bundle [::xserv::getBundleName $xservname]
930   if {$bundle != ""} {
931     set xservname $bundle
932   }
933
934   array set defaultImpl [list]
935   if {[info exists ::xserv::currentImplementations($xservname)]} {
936     array set current [set ::xserv::currentImplementations($xservname)]
937     if {[info exists current($group)]} {
938       array set defaultImpl $current($group)
939     } elseif {[info exists current("")]} {
940       array set defaultImpl $current("")
941     }
942   }
943
944   set implList [::xserv::getImplementationsOf $xservname]
945
946   if {![::xserv::isBundle $xservname]} {
947     set mandatory [::xserv::mandatoryArgsOf $xservname]
948     if {[llength $mandatory] == 1} {
949       lappend implList "* Other *"
950     }
951   }
952
953   if {[info exists defaultImpl(-name)]} {
954     if {[catch {set theImpl [  listpick -p "Choose an implementation for\
                  $xservname"  -L [list $defaultImpl(-name)] $implList  ]}]} {
958       return [list]
959     }
960   } else {
961     if {[catch {set theImpl [  listpick -p "Choose an implementation for\
                                              $xservname"  $implList  ]}]} {
965       return [list]
966     }
967   }
968   return\
          [::xserv::chooseImplementationFor $xservname [list $theImpl] $group]
969 }
970
```

xserv::editHelpers (proc)

The `xserv::editHelpers` procedure allows the user to choose an XSERV and then an implementation of this XSERV. It can be used to let the user configure all declared XSERVs (like in Preferences->Helper Applications).

If there is more than one category of services, this procedure first asks the user to choose a category of services, and then displays only the services in this category.

Services which belong to no category are put in a virtual `* no category *` category.

```
971 proc ::xserv::editHelpers {{group ""}} {
```

{*group*} is the group for which the implementation of a service will be chosen. It defaults

to "", the default group.

```
global ::xserv::categories
```

First, all services are put in the list of services with no category, and the list of all categories which contain services is built.

```
set noCat [::xserv::listOfServices]
if {[info exists ::xserv::categories]} {
  foreach cat [array names ::xserv::categories] {
    set avails [list]
    foreach serv [set ::xserv::categories($cat)] {
      if {[lsearch -exact $noCat $serv] != -1} {
        lappend avails $serv
      }
    }
    if {[llength $avails] > 0} {
      set editCats($cat) $avails
    }
  }
```

Then, services which belong to a category are removed from the list of uncategorized services.

```
  if {[info exists editCats]} {
    foreach cat [array names editCats] {
      foreach serv $editCats($cat) {
        set idx [lsearch -exact $noCat $serv]
        if {$idx != -1} {
          set noCat [lreplace $noCat $idx $idx]
        }
      }
    }
```

If there are services with no category, we add a special category for them in the list, labelled `* no category *`.

```
    if {[llength $noCat] > 0} {
      set editCats(*\ no\ category\ *) $noCat
    }
  }
}

if {[info exists editCats]} {
  set theCats [array names editCats]
  if {[llength $theCats] > 1} {
```

If there is more than one category, let the user choose a category

```
    while 1 {
      set status [catch {set theCat [  listpick -p "Choose a category\
                          of services"  [lsort -dictionary $theCats]  ]}]
      if {$status} {
        return
```

```
1011          } else {
1012            set status [catch {set theXSERV [  listpick -p "Choose a\
                   service to edit"  [lsort -dictionary $editCats($theCat)]  ]}]
1016            if {!$status} {
1017              return [list $theXSERV \
                           [::xserv::selectImplementationFor $theXSERV $group]]
1019            }
1020          }
1021        }
1022      }
1023    } else {
1024      if {[llength $noCat] == 0} {
1025        alertnote "There are no declared services yet."
1026        return
1027      }
```

If there is only one category, don't show the category dialog

```
1028      set status [catch {set theXSERV [  listpick -p "Choose a service to\
                                          edit"  [lsort -dictionary $noCat]  ]}]
1032      if {$status} {
1033        return
1034      } else {
1035        return [list $theXSERV \
                           [::xserv::selectImplementationFor $theXSERV $group]]
1037      }
1038    }
1039 }
1040
```

xserv::setExternalHelpers (proc)

The `setExternalHelpers` procedure is just a wrapper around `xserv::editHelpers` which is the callback of the "Set external helpers" item added to the Config menu.

```
1041 proc setExternalHelpers {} {
1042   ::xserv::editHelpers
1043 }
1044
```

3.9 Invoking XSERVs

xserv::invoke (proc)

The `xserv::invoke` procedure asks an XSERV to perform its task through its current implementation (as chosen by the default group).

The ⟨*interact*⟩ parameter should be set to `-foreground` if the user is expected to interact with the application, or to `-background` if the application should operate silently.

The arguments of the XSERV must be in the form {*-key*} {*value*}, where {*key*} is the name of a formal parameter of the XSERV, and {*value*} is the actual value of the parameter. For instance, to typeset the file `hello.tex` with the `latex` format, passing option `--src` to the TEX implementation, we can write:

```
xserv::invoke -foreground tex -filename hello.tex -format latex -options --src
```

From version 1.3, the leading - in front of the keys can no longer be omitted.

If a parameter is not set, its default value (as declared in the XSERV) is used. If no default value is declared, this is an error (just like for a Tcl proc).

```
proc ::xserv::invoke {interact xservname args} {
  set invocation [list ::xserv::invokeForGroup {} $interact $xservname]
  eval [concat $invocation $args]
}
```

xserv::invokeForGroup (proc)

The `xserv::invokeForGroup` procedure asks an XSERV to perform its task through the current implementation chosen by a group.

```
proc ::xserv::invokeForGroup {group interact xservname args} {
  global ::xserv::services
  global ::xserv::currentImplementations
  global ::xserv::endExecHooks

  switch -- "$interact" {
    "-foreground"  {set interaction 1}
    "-background"  {set interaction 0}
    default {
      error "Unknown value \"$interact\" for 'interact' parameter"
    }
  }

  if {![info exists ::xserv::services($xservname)]} {
    error "Undeclared service \"$xservname\""
  }
  if {![info exists ::xserv::currentImplementations($xservname)]} {
    array set current [list]
  } else {
    array set current [set ::xserv::currentImplementations($xservname)]
  }
  if {![info exists current($group)]} {
    array set theImpl [::xserv::selectImplementationFor $xservname $group]
    if {[array size theImpl] == 0} {
      error "Cancel"
    }
  } else {
    array set theImpl "$current($group)"
  }

  # For now, we don't know how to handle an implementation choice without
  # a name in it.
  if {![info exists theImpl(-name)]} {
    error "No implementation name in [array get theImpl]"
  }

  array set theXserv [set ::xserv::services($xservname)]
  if {![info exists theXserv(implementations)]} {
    array set theImpls [list]
  } else {
```

```
1090      array set theImpls $theXserv(implementations)
1091    }
1092    if {![info exists theImpls($theImpl(-name))]} {
1093      set msg "Implementation \"$theImpl(-name)\" "
1094      append msg "does not support service \"$xservname\""
1095      error $msg
1096    }
1097
1098    array set imp $theImpls($theImpl(-name))
1099    if {[info exists imp(-mode)]} {
```

If a validation procedure exists for this invocation mode, call it to validate the chosen implementation.

```
1100      if {[info commands ::xserv::validateImpChoice$imp(-mode)] != ""} {
1101        set validated [  ::xserv::validateImpChoice$imp(-mode) \
                              [array get theImpl]  $theImpls($theImpl(-name)) ]
1106        if {[llength $validated] == 0} {
1107          error "Could not execute $theImpl(name) for $xservname"
1108        } else {
1109          unset theImpl
1110          array set theImpl $validated
1111          ::xserv::chooseImplementationFor $xservname $validated $group
1112        }
1113      }
1114    } else {
1115      error "No invocation mode for $theImpl(-name)."
1116    }
1117
1118    array set imp [array get theImpl]
1119    set imp(xservName) $xservname
1120  #   set imp(xservImpl) $theImpl(-name)
1121
1122    # Get the list of formal parameters from the declaration of the XSERV
1123    set formalargs $theXserv(args)
1124
1125    # Get the effective arguments from the "args" trailing parameters
1126    if {[llength $args] % 2 != 0} {
1127      error {Malformed parameter list. Must be [key value]*}
1128    }
1129    array set effectiveargs $args
1130    if {[llength [array names effectiveargs]]  !=\
                                  [llength [array names effectiveargs -*]]} {
1132      error {Vince said keys must have a leading '-'}
1133    }
1134
1135    # Remove leading '-' in parameter names so that the driver
1136    # can access them with '$params(<param name>)'
1137    foreach name [array names effectiveargs] {
1138      set effectiveargs([string range $name 1 end]) $effectiveargs($name)
1139      unset effectiveargs($name)
1140    }
1141
1142    # Check for unknown arguments
```

```
1143  foreach name [array names effectiveargs] {
1144    if {[lsearch -exact $formalargs $name] < 0} {
1145      if {[lsearch -regexp $formalargs [list $name *]] < 0} {
1146        error "Unknown parameter \"$name\" in \"$xservname\"."
1147      }
1148    }
1149  }
1150  # Add default values for omitted parameters
1151  foreach arg $formalargs {
1152    set argname [lindex $arg 0]
1153    if {![info exists effectiveargs($argname)]} {
1154      if {[llength $arg] > 1} {
1155        set effectiveargs($argname) [lindex $arg 1]
1156      } else {
1157        error "No value for parameter \"$argname\" in \"$xservname\"."
1158      }
1159    }
1160  }
1161
1162  set effectiveargs(xservInteraction) $interaction
1163  # We use a temporary proc to insulate the driver
1164  # script from our local variables.
1165  set procText "\n  array set params \$args;"
1166  append procText $imp(-driver)
1167  eval proc ::xserv::tmpProc {args} {$procText}
1168
1169  if {[info commands ::xserv::execute$imp(-mode)] != ""} {
1170    set result [ ::xserv::execute$imp(-mode)  [array get imp] \
                                                    [array get effectiveargs] ]
1175    return $result
1176  } else {
1177    error "No handler for \"$imp(-mode)\" invocation mode."
1178  }
1179 }
1180
```

xserv::execEndExecHooks (proc)

The `xserv::execEndExecHooks` procedure calls all end of execution hooks registered for the service. Its code has been factored out of `xserv::invokeForGroup` so that it can be reused by different execution procedure as new invocation modes are added to xserv.

```
1181 proc ::xserv::execEndExecHooks {imp effectiveargs result} {
```

{*imp*} is a key-value list which describes the implementation. {*effectiveargs*} is a key-value list which gives the values of the arguments of the invocation. {*result*} contains the result of the invocation of the service.

```
1182  global ::xserv::endExecHooks
1183
1184  array set theImpl $imp
1185  set xservname [set theImpl(xservName)]
1186  if {  [info exists ::xserv::endExecHooks($xservname)]
1187    && ([llength [set ::xserv::endExecHooks($xservname)]] > 0)} {
1188    foreach p [set ::xserv::endExecHooks($xservname)] {
1189      eval [list $p $imp $effectiveargs $result]
```

```
1190      }
1191    }
1192 }
1193
```

`xserv::addEndExecHook` (proc)

The `xserv::addEndExecHook` procedure adds a procedure to the list of procedures to call after each invocation of an XSERV.

`xservname` is the name of the XSERV, and `proc` is the name of the procedure to call. This procedure will be called with four arguments:

1. a list which describes the implementation of the XSERV used for this invocation;
2. the list of paramaters of the invocation.
3. the result of the invocation;

The first two parameters are key-value lists suitable for use with `array set`.

```
1194 proc ::xserv::addEndExecHook {xservname proc} {
1195   global ::xserv::endExecHooks
1196   global ::xserv::services
1197
1198   if {![info exists ::xserv::services($xservname)]} {
1199     error "Undeclared service \"$xservname\""
1200   }
1201   if {![info exists ::xserv::endExecHooks($xservname)]
1202     || [lsearch -exact "$proc" [set ::xserv::endExecHooks($xservname)]]\
                                                                    == -1} {
1203     lappend ::xserv::endExecHooks($xservname) $proc
1204   }
1205 }
1206
```

`xserv:`
`:removeEndExecHook` (proc)

The `xserv::removeEndExecHook` procedure removes a procedure from the list of procedures to call after each invocation of an XSERV.

`xservname` is the name of the XSERV, and `proc` is the name of the procedure to remove. If `proc` is omitted, the list of procedures to call will be made empty.

If the list is empty or does not contain `proc`, `removeEndExecHook` will do nothing.

```
1207 proc ::xserv::removeEndExecHook {xservname {proc ""}} {
1208   global ::xserv::endExecHooks
1209   global ::xserv::services
1210
1211   if {![info exists ::xserv::services($xservname)]} {
1212     error "Undeclared service \"$xservname\""
1213   }
1214
1215   if {$proc == ""} {
1216     set ::xserv::endExecHooks($xservname) [list]
1217   } elseif {[info exists ::xserv::endExecHooks($xservname)]} {
```

```
1218    set idx\
              [lsearch -exact "$proc" [set ::xserv::endExecHooks($xservname)]]
1219    if {$idx != -1} {
1220      set ::xserv::endExecHooks($xservname)\
                    [ lreplace ::xserv::endExecHooks($xservname) $idx $idx ]
1223    }
1224  }
1225 }
1226
```

xserv::executeApp (proc)

The xserv::executeApp procedure executes an application. This is one of the possible final steps in the invocation of an XSERV. It is used when the incocation mode is App.

implArray is the array-set like list which describes the implementation to use.

paramArray is the array-set like list which contains the values of the parameters of the invocation.

```
1227 proc ::xserv::executeApp {implArray paramArray} {
1228   array set imp $implArray
1229   array set params $paramArray
1230   set app "$imp(-path)"
1231   # the target is the short name of the application
1232   # without the ".app" extension in OS X
1233   regsub {\.app$} [file tail $app] {} params(xservTarget)
1234   set cmd [list launch]
1235   if {$params(xservInteraction)} {
1236     lappend cmd "-f"
1237   }
1238   lappend cmd $app
1239   if {[catch {eval $cmd}]} {
1240     error "Could not launch $app"
1241   }
1242   set result [eval ::xserv::tmpProc [array get params]]
1243   ::xserv::execEndExecHooks $implArray $paramArray $result
1244   return $result
1245 }
1246
```

xserv::executeShell (proc)

The xserv::executeShell procedure executes a shell and sends the result of the driver script to its standard input. This is one of the possible final steps in the invocation of an XSERV. It is used when the incocation mode is Shell.

The result of the driver script is interpreted as a list of words. Each word in this list is processed to escape the spaces it may contain before sending it to the standard input of the shell.

implArray is the array-set like list which describes the implementation to use.

paramArray is the array-set like list which contains the values of the parameters of the invocation.

```
1247 proc ::xserv::executeShell {implArray paramArray} {
1248   array set imp $implArray
1249   array set params $paramArray
```

```
1250   if {[info exists imp(-progs)]} {
1251     # For each requested program, make the absolute path
1252     # to the program available in the params array.
1253     array set theProgs $imp(-progs)
1254     foreach p [array names theProgs] {
1255       set params(xserv-$p) $theProgs($p)
1256     }
1257     set paramArray [array get params]
1258   }
1259   set cmd "[eval ::xserv::tmpProc $paramArray]"
1260   set cmdline ""
1261   foreach word $cmd {
1262     regsub -all " " $word "\\ " word
1263     append cmdline " $word"
1264   }
1265   if {!$params(xservInteraction)} {
1266     set result [exec $imp(-shell) << $cmdline &]
1267   } else {
1268     set result [exec $imp(-shell) << $cmdline]
1269   }
1270   ::xserv::execEndExecHooks $implArray $paramArray $result
1271   return $result
1272 }
1273
```

xserv::executeInSh (proc)

The `xserv::executeInSh` procedure executes a shell and writes the result of the driver script to its standard input. A window in `InSh` *Alpha* mode is used to let the user interact with the shell.

This is one of the possible final steps in the invocation of an XSERV. It is used when the incocation mode is `InSh`.

The result of the driver script is interpreted as a list of words. Each word in this list is processed to escape the spaces it may contain before sending it to the standard input of the shell.

`implArray` is the array-set like list which describes the implementation to use.

`paramArray` is the array-set like list which contains the values of the parameters of the invocation.

```
1274 proc ::xserv::executeInSh {implArray paramArray} {
1275   array set imp $implArray
1276   array set params $paramArray
1277   if {!$params(xservInteraction)} {
1278     # Background invocation for an interactive shell
1279     # => use a non-interactive shell
1280     return [::xserv::executeShell $implArray $paramArray]
1281   }
1282   if {[info exists imp(-progs)]} {
1283     # For each requested program, make the absolute path
1284     # to the program available in the params array.
1285     array set theProgs $imp(-progs)
1286     foreach p [array names theProgs] {
1287       set params(xserv-$p) $theProgs($p)
```

```
1288    }
1289    set paramArray [array get params]
1290  }
1291  set cmd "[eval ::xserv::tmpProc $paramArray]"
1292  set cmdline ""
1293  foreach word $cmd {
1294    regsub -all " " $word "\\ " word
1295    append cmdline " $word"
1296  }
1297  set wname "$imp(xservName)"
1298  set ignored ""
1299  if {[info exists imp(-ignore)]} {
1300    set ignored $imp(-ignore)
1301  }
1302  # The default InSh mode is "ioe": the shell window handles
1303  # the input, output and error streams.
1304  set iomode "ioe"
1305  if {[info exists imp(-ioMode)]} {
1306    set iomode $imp(-ioMode)
1307  }
1308  set inchannel\
                [InSh::createShell $wname "$imp(-shell) -i" $ignored $iomode]
1309  # Ask for calling the end-exec hooks when the socket is closed.
1310  InSh::addCloseHook $wname\
                    [list ::xserv::execEndExecHooks  $implArray $paramArray]
1312  puts $inchannel "$cmdline;exit"
1313 }
1314
```

xserv::executeExec (proc)

The xserv::executeExec procedure executes the result of the driver script with the Tclexec command. This is one of the possible final steps in the invocation of an XSERV. It is used when the incocation mode is Exec.

The result of the driver script is interpreted as a list of words. Each word in this list is processed to escape the spaces it may contain before passing it to the exec command.

implArray is the array-set like list which describes the implementation to use.

paramArray is the array-set like list which contains the values of the parameters of the invocation.

```
1315 proc ::xserv::executeExec {implArray paramArray} {
1316   array set imp $implArray
1317   array set params $paramArray
1318   if {[info exists imp(-progs)]} {
1319     # For each requested program, make the absolute path
1320     # to the program available in the params array.
1321     array set theProgs $imp(-progs)
1322     foreach p [array names theProgs] {
1323       set params(xserv-$p) $theProgs($p)
1324     }
1325     set paramArray [array get params]
1326   }
1327   set cmd "[eval ::xserv::tmpProc $paramArray]"
1328   set cmdline ""
```

```
  foreach word $cmd {
    regsub -all " " $word "\\ " word
    append cmdline " $word"
  }

  set cmd "exec $cmdline"
  if {!$params(xservInteraction)} {
    append cmd " &"
  }
  catch {eval $cmd} result
  ::xserv::execEndExecHooks $implArray $paramArray $result
  return $result
}

```

xserv::executeAlpha (proc)

The `xserv::executeAlpha` procedure executes the result of the driver script with the Tclinterpreter of *Alpha*. This is one of the possible final steps in the invocation of an XSERV. It is used when the incocation mode is `Alpha`.

The driver script is interpreted by Alpha in its own context, as if it was the body of a procedure.

`implArray` is the array-set like list which describes the implementation to use.

`paramArray` is the array-set like list which contains the values of the parameters of the invocation.

```
proc ::xserv::executeAlpha {implArray paramArray} {
  array set imp $implArray
  array set params $paramArray
  set result [eval ::xserv::tmpProc $paramArray]
  ::xserv::execEndExecHooks $implArray $paramArray $result
  return $result
}

```

</tcl>

4 User help file

<*help>

```
-->
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <title>Xserv Help</title>
  <meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
  <LINK REV="MADE" HREF="mailto:Frederic.Boulanger@supelec.fr">
  <META NAME="author" CONTENT="Fr&eacute;d&eacute;ric Boulanger">
  <META NAME="date" CONTENT="2003-06-24">
</head>
```

```
<body>

<h1 align="center">Xserv help</h1>

<p>Version 1.3</p>
<p><a href="mailto:Frederic.Boulanger@supelec.fr">
   Fr&eacute;d&eacute;ric Boulanger</a></p>

<h2>Introduction</h2>

<p>Xserv is an AlphaTcl package which introduces a new way to manage external
applications. Each operation that can be done by an application is called a
service. For instance, we may have a "viewURL" service, the effect
of which is to display the contents of an URL. There may be several
implementations of this service: one using "Internet Explorer", another
using "Safari" or yet another using "Netscape".</p>

<p>Xserv allows packages and modes to declare services and to use them in a
uniform way, independent of the implementation used to provide the
service.</p>

<p>Xserv interacts with you only for choosing an implementation for a
service. If a service is invoked and no implementation has been chosen for
it yet, Xserv will display a dialog containing the list of all known
implementations of the service. Once an implementation has been chosen, it
is remembered until you decide to choose another implementation.</p>


<h2>Xserv and Helper applications</h2>

<p>Since Xserv is new, most of the code in AlphaTcl does not use it yet. Xserv
comes with some files to use with the "Smarter Source" extension to modify
existing modes or packages so that they use Xserv. If you want to use them,
you must activate the "Smarter Source" feature in "Config->Global
Setup->Features", and put the files (latexComm+.tcl, htmlProcs+.tcl and
diffMode+.tcl) in your "Tcl Extensions" folder. The "Tcl Extensions" folder
is set in the "Files" page of the dialog displayed by
"Config->Preferences->Input - Output Preferences". You will have to restart
Alpha to make sure that these files are used.</p>

<p>However, many parts of AlphaTcl don’t use Xserv. To set which application
to use for different tasks for these parts of AlphaTcl, use the "Helper
Applications" item in "Config --> Global Setup".</p>

<p>If your settings in Xserv seem to have no effect (for instance, you choose
Safari for the viewURL service, and Internet Explorer is opened when you try
to view an URL), this is probably because Xserv is not used by some part of
AlphaTcl. In this case, use the "Helper Applications" item to set the
application you want to use.</p>


<h2>Choosing which application to use for what</h2>
```

Xserv adds a "Set External Helpers" item in he "Config->Global Setup" menu. This item allows you to choose an implementation for all declared services. Since it may be difficult to find a service in a very long list, services are grouped into categories. If there is more than one category of service, you will have to select a category of services first. Categories are defined by the developers when they declare services and should have inspired names... For instance, the "viewURL" service could be put in a "WWW" category.

Note: This dialog shows only the categories, services and implementations that are known to Xserv. If a mode or package declares services but has never been loaded since Xserv has been installed, those services and their implementations won’t appear in the dialog.

Once you have selected a category, you should see the list of the services in this category. Select the one for which you want to change or set the implementation. You should then see the list of all known implementations of this service. The name of an implementation is generally the name of the application used to provide the service.

If you make an error or choose an implementation which is not the one you expected, you can always go back to this dialog and choose another implementation for the service.

<h2>Using an application which is not in the list</h2>

If you don’t see the name of your favorite application in the list of the implementations of a service, it may be that the implementation which uses this application has a weird name, or more probably that no one has told Xserv how to use this application for the service.

If you can write Tcl code and read the programmer’s manual of Xserv, you can register your application as an implementation of the service (tell Xserv how to use this application).

However, for simple services, Xserv can learn how to use your application if you know enough about Apple Events or command lines. If the service is simple enough, you will see an item labelled "* Other *" in the list of the implementations of the service. Choosing this item will make a dialog appear to let you enter the information needed by Xserv to use your application. From the top pop-up menu of this dialog, you choose whether your application understands Apple Events (like Mac OS applications), or receives arguments from a command line (like Unix programs).

<h3>Using an Apple Event driven application</h3>

For Apple Events applications, you must give the name of the application in the "Application" text field. If you prefer, you can give the creator code of the application between single quote (for instance, ’ttxt’ for SimpleText or TextEdit). Then, you must give the class and code of the Apple Event that will be sent to the application. The default is an Apple Event of class aevt and of code odoc, which is the standard Apple Event to ask an application to open a document.

If you want to print a document, you can try aevt and pdoc. To open a URL, the Apple Event has class GURL and code GURL or WWW! and OURL (for Internet Explorer).

The last pop-up menu allows to choose the type of the argument: for an odoc Apple Event, the type is "file", which means that the Apple Event will carry a reference to a file on your disk. For a GURL Apple Event, the type should be set to "text" since the URL is just a piece of text.

<h3>Using a command-line program</h3>

For command line programs, you must give the name of the program (for instance "gs" for ghostview).

The mode is the way the program will be executed. The default is "InSh" which will execute the program in an interactive window: you will see the output of the program and you will be able to type text if the program needs some input. "Shell" and "Exec" are two non-interactive mode: the program will be executed but you won't see anything until Alpha gets the result and does something with it. In "Exec" mode, the program is executed directly by the operating system, while in "Shell" mode, the program is executed by a shell program (like the one that reads what you type in a terminal).

The last item in the dialog allows you to give the general form of the command line. The default is to use the name of the program (represented by the "<prog>" string), followed by the argument (represented by the "$params(...)" string, in which the three dots are replaced by the actual name of the argument). For instance, if you want to use the program with the "-verbose" option, and if the argument must be prefixed by "-input=", you should set the command line to:

<pre>
     <prog> -verbose -input=$params(...)
</pre>

where "..." is the name of the argument, as shown in the default value of the field.

When you have entered all the necessary information, click "OK". This will create and select a new implementation of the service. Such implementations are called "generic implementations" since they use simple and generic mechanisms to interact with an application. Their name is always in the form "generic-<prog>", where <prog> is the name of the program or application used by the implementation.

<h3>Deleting generic implementations</h3>

If a generic implementation doesn't work or is no longer needed, it can be deleted using the "Delete Generic Implementation" item in the "Config->Global Setup" menu.

```
</body>
</html>
```

</help>

5 Examples

```
⟨∗examples⟩

###
## Examples of use of the xserv package in various modes
###

###
## For TeX mode
###
::xserv::declare viewDVI {Display a DVI file} file
::xserv::declare viewPS {Display a PostScript file} file
::xserv::declare viewPDF {Display a PDF file} file

::xserv::declare printDVI {Print a DVI file} file
::xserv::declare printPS {Print a PostScript file} file
::xserv::declare printPDF {Print a PDF file} file

::xserv::declare dvips {Convert a DVI file to PostScript} file\
                                                    {options ""}
::xserv::declare dvipdf {Convert a DVI file to PDF} file
::xserv::declare distillPS {Convert a PS file to PDF} file\
                                                    {options ""}

::xserv::declare bibtex\
               {Build a bibliography from a LaTeX aux file}  file\
                                                    {options ""}
::xserv::declare makeindex\
           {Build an index from a LaTeX idx file}  file {style ""}\
                                                    {options ""}
::xserv::declare makeglossary\
         {Build a glossary from a LaTeX glo file}  file {style ""}\
                                                    {options ""}

::xserv::declare tex {Typeset a file with TeX}  file\
                                      {format "latex"} {options ""}
::xserv::declare etex {Typeset a file with eTeX}  file\
                                     {format "elatex"} {options ""}
::xserv::declare pdftex {Typeset a file with pdfTeX}  file\
                                   {format "pdflatex"} {options ""}
::xserv::declare pdfetex {Typeset a file with pdfeTeX}  file\
                                  {format "pdfelatex"} {options ""}

###
## Set up categories for all these services
```

```
###
::xserv::addToCategory TeX tex etex pdftex pdfetex bibtex\
                                          makeindex makeglossary
::xserv::addToCategory DVI viewDVI printDVI dvips dvipdf
::xserv::addToCategory PostScript viewPS printPS distillPS\
                                                 dvips dvipdf
::xserv::addToCategory PDF viewPDF printPDF distillPS dvipdf

# Implementations of the APIs
::xserv::register viewPDF Acrobat -sig CARO -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register viewPDF {Apple Preview} -sig prvw -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register viewPDF {Finder choice} -sig MACS -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register viewPDF {PDFViewer} -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
} -mode App

::xserv::register bibtex {CMacTeX < 4} -sig CMTu -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register bibtex BibTeX -sig Vbib -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register dvips {CMacTeX < 4} -sig CMT1 -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register dvips OzTeX -sig OzDP -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register dvipdf {CMacTeX < 4} -sig CMTb -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register makeindex {CMacTeX < 4} -sig CMTt -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register makeindex MakeIndex -sig RZMI -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

#
```

```
# The default is to implement make glossary with makeindex
#
::xserv::register makeglossary {
  ::xserv::invoke $params(interaction) makeindex  -file\
                    $params(file) -style $params(style) -options\
                                                $params(options)
}

::xserv::register printDVI CMacTeX -sig CMT8 -driver {
  tclAE::send $params(xservTarget) aevt pdoc  ----\
                                [tclAE::build::alis $params(file)]
}

::xserv::register printDVI OzTeX -sig OTEX -driver {
  tclAE::send $params(xservTarget) aevt pdoc  ----\
                                [tclAE::build::alis $params(file)]
}

::xserv::register tex {CMacTeX < 4} -sig *XeT -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register tex OzTeX -sig OTEX -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register viewDVI {CMacTeX < 4} -sig CMT8 -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register viewDVI OzTeX -sig OTEX -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

::xserv::register pdftex {CMacTeX < 4} -sig pXeT -driver {
  sendOpenEvent noReply $params(xservTarget) $params(file)
}

#
# New CMacTeX 4 API
#
::xserv::register pdftex {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget)  "pdftex\
                $params(options) &$params(format)" $params(file)
}

::xserv::register pdfetex {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget)  "pdfetex\
                $params(options) &$params(format)" $params(file)
}

::xserv::register tex {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget)  "tex\
                $params(options) &$params(format)" $params(file)
```

```
}

::xserv::register etex {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget)  "etex\
                $params(options) &$params(format)" $params(file)
}

::xserv::register makeindex {CMacTeX >= 4} -sig *XeT -driver {
  if {$params(style) == ""} {
    TeX::buildNewCMacTeXAE $params(xservTarget)  "makeindex\
                                  $params(options)" $params(file)
  } else {
    TeX::buildNewCMacTeXAE $params(xservTarget)  "makeindex\
              $params(options) -s $params(style)" $params(file)
  }
}

::xserv::register dvipdf {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget) dvipdfm\
                                                  $params(file)
}

::xserv::register bibtex {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget)  "bibtex\
                                  $params(options)" $params(file)
}

::xserv::register dvips {CMacTeX >= 4} -sig *XeT -driver {
  TeX::buildNewCMacTeXAE $params(xservTarget)  "dvips\
                                  $params(options)" $params(file)
}


#
# teTeX
#
::xserv::register pdftex teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-pdftex) -fmt=$params(format)
  lappend cmd $params(options) [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {pdftex} -ioMode "o"

::xserv::register pdfetex teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-pdfetex) -fmt=$params(format)
  lappend $params(options) [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {pdfetex} -ioMode "o"

::xserv::register tex teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-tex) -fmt=$params(format)
  lappend cmd $params(options) [file tail $params(file)]
```

```
  return $cmd
} -mode InSh -shell sh -progs {tex} -ioMode "o"

::xserv::register etex teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-etex) -fmt=$params(format)
  lappend cmd $params(options) [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {etex} -ioMode "o"

::xserv::register makeindex teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-makeindex) $params(options)
  if {$params(style) != ""} {
    lappend cmd -s $params(style)
  }
  lappend cmd [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {makeindex} -ioMode "oe"

::xserv::register dvipdf teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-dvipdfm) [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {dvipdfm} -ioMode "o"

::xserv::register bibtex teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-bibtex) $params(options)\
                                        [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {bibtex} -ioMode "o"

::xserv::register dvips teTeX -driver {
  set cmd [list cd [file dirname $params(file)] \;]
  lappend cmd $params(xserv-dvips) $params(options)\
                                        [file tail $params(file)]
  return $cmd
} -mode InSh -shell sh -progs {dvips} -ioMode "o"


#
# APIs for TeXtures
#
# (not tested, but may be a starting point for a working\
                                                 implementation)
#
::xserv::declare openTeXtureConnection  "Begin to work with a\
                                          file in TeXtures" file

::xserv::declare closeTeXtureConnection  "Stop working with a\
                                          file in TeXture" jobID

```

```
::xserv::declare synchronizeTeXture  "Synchronize with\
                                          TeXtures" jobID position

::xserv::declare getTeXtureFormats "Get the formats available\
                                                    to TeXture"

::xserv::declare handleTeXtureGetText  "Reply to TeXture\
                                    requests for text" jobID text

###
# Put all TeXtures services in a "TeXtures" bundle
###
::xserv::declareBundle TeXtures {TeXtures protocol} \
                    openTeXtureConnection  closeTeXtureConnection \
       synchronizeTeXture  getTeXtureFormats  handleTeXtureGetText

# Put the "TeXtures" bundle in the "TeX" category
::xserv::addToCategory TeX TeXtures

# Implementation of the TeXture specific APIs
::xserv::register openTeXtureConnection Texture -sig *TEX\
                                                    -driver {
  tclAE::build::resultData $params(xservTarget) BSRs Begi \
                           ---- [tclAE::build::alis $params(file)]
}

::xserv::register closeTeXtureConnection Texture -sig *TEX\
                                                    -driver {
  tclAE::build::resultData $params(xservTarget) BSRs Disc Jobi\
                                                  $params(jobID)
}

::xserv::register synchronizeTeXture Texture -sig *TEX -driver\
                                                        {
  tclAE::build::resultData $params(xservTarget) BSRs FFoc \
                        long $params(position) Jobi $params(jobID)
}

::xserv::register getTeXtureFormats Texture -sig *TEX -driver\
                                                        {
  tclAE::build::resultData $params(xservTarget) BSRs Info Fmts\
                                                     long(0)
}

::xserv::register handleTeXtureGetText Texture -sig *TEX\
                                                    -driver {
  tclAE::build::resultData -t 1200 $params(xservTarget) BSRs\
             TTeX  TEXT [tclAE::build::TEXT "$params(text)"] Jobi\
                                                  $params(jobID)
}

##
# Replacement for TeX mode procs
#
```

```
# (waiting for TeX mode to switch to xserv)
##
proc pdflatexTEXFile {filename} {
  xserv::invoke pdftex -file $filename -format pdflatex
}

proc bibtexAUXFile {filename} {
  xserv::invoke bibtex -file $filename
}

proc makeindexIDXFile {filename} {
  xserv::invoke makeindex -file $filename
}

proc makeindexGLOFile {filename} {
  ::xserv::invoke -foreground makeglossary -file $filename
}

proc viewDVIFile {filename} {
  ::xserv::invoke -foreground viewDVI -file $filename
}

proc viewPDFFile {filename} {
  ::xserv::invoke -foreground viewPDF -file $filename
}

proc printDVIFile {filename} {
  ::xserv::invoke -foreground printDVI -file $filename
}

proc dvipsDVIFile {filename} {
  ::xserv::invoke -foreground dvips -file $filename
}

proc dvipdfDVIFile {filename} {
  ::xserv::invoke -foreground dvipdf -file $filename
}

proc viewPSFile {filename} {
  if {[catch\
        {::xserv::invoke -foreground viewPS -file $filename}]} {
    status::msg "View aborted."
  }
}

proc printPSFile {filename} {
  ::xserv::invoke -foreground printPS -file $filename
}

proc distillPSFile {filename} {
  ::xserv::invoke -foreground distillPS -file $filename
}

# Send a command line to CMacTeX >= 4.0
```

```
proc TeX::buildNewCMacTeXAE {target command filename} {
  return [tclAE::send -p $target CMTX exec  ----\
          [tclAE::build::TEXT "$command [file tail $filename]"] \
                                      dest [tclAE::build::alis\
                  "[file dirname $filename][file separator]"]  ]
}

proc TeX::effectiveFormat {} {
  global TeXmodeVars TeX::FormatOptions TeX::TypesetFile

  set TeXprogram   \
                  [string tolower $TeXmodeVars(nameOfTeXProgram)]
  set TeXformat     $TeXmodeVars(nameOfTeXFormat)
  set formatOption  $TeXmodeVars(formatShouldBe)
  set formatOptions $TeXmodeVars(availableTeXFormats)

  set format ""
  # If the format option is "0" we ignore anything in the\
                                                    Format pref.
  if {$formatOption != "0"} {
    if {![string length $TeXformat]} {
      # Format names are generally auto-adjusted when the\
                                                      window is
      # activated, but maybe we're being called without the\
                                                        window
      # being open.
      set formatName\
         [lindex [TeX::getFormatName [set TeX::TypesetFile]] 0]
      # Make sure that it's a valid option.
      if {[lsearch $formatOptions $formatName] != "-1"} {
        set TeXformat $TeXmodeVars(nameOfTeXFormat)
      }
      if {![string length $TeXformat]} {
        # We still don't have one.
        regsub -all {\(-\)*} $formatOptions "-" formatOptions
        set pArgs\
            [list  "Please choose a format" "LaTeX" "options:"]
        if {[catch {eval prompt $pArgs $formatOptions}\
                                                 formatName]} {
          status::errorMsg "Cancelled."
        }
        set TeXformat $TeXmodeVars(nameOfTeXFormat)
      }
      set TeXmodeVars(nameOfTeXFormat) $formatName
    }
    if {$formatOption == "1"} {
      # If the format names are fixed, they do not depend
      # on the name of the TeX program, so their prefix is\
                                                        empty.
      set format "${TeXformat}"
    } elseif {$formatOption == "2"} {
      # Guess the effective name of the format (e.g.\
                                                    pdfelatex)
```

```
1743      # from the generic name of the format (e.g. latex) and\
                                                                the
1744      # name of the TeX program (e.g. pdfetex).  For this,\
                                                                  take the
1745      # prefix of "tex" in the name of the TeX program. \
                                                                  Warning:
1746      # this does not work with the usual names of the Omega
1747      # formats (omega and lambda).
1748      if {![regexp -nocase -- {(.*)tex} $TeXprogram dummy\
                                                                 prefix]} {
1749        alertnote "Couldn't not determine prefix of\
                                                          \"$TeXprogram\""
1750        return ""
1751      }
1752      regsub -nocase -- {(la)?(tex)} ${TeXformat} "\\1\\2"\
                                                                  TeXformat
1753      set format "${prefix}${TeXformat}"
1754    } else {
1755      status::errorMsg "Unknown formatting option:\
                [lindex [set TeX::FormatOptions] $formatOption]"
1756    }
1757  }
1758  return $format
1759 }
1760
1761 proc TeX::buildCMacTeXcommand {} {
1762  set cmdline [string tolower $TeXmodeVars(nameOfTeXProgram)]
1763  set format "[TeX::effectiveFormat]"
1764  if {$format != ""} {
1765    append cmdline " &$format"
1766  }
1767  return cmdline
1768 }
1769
1770 # There are no longer tex, pdftex and so on TeX commands.  You\
                                                           just typeset a
1771 # file, what amounts to asking the current TeX program to\
                                                          compile the file
1772 # using the current TeX format which is either set when the\
                                                           file is brought
1773 # to the front or manually selected.
1774 proc TeX::typesetFile {filename {bg 0}} {
1775  global TeXmodeVars TeX::TypesetFile
1776  global showTeXLog
1777
1778  if {$filename == ""} {
1779    set filename [getfile "Choose a file to typeset:"]
1780  }
1781  TeX::typesetFirstMessage
1782  # Just so other code knows what we're dealing with.
1783  set TeX::TypesetFile $filename
1784
1785  if {$bg == 0} {
1786    set interact "-foreground"
```

```
  } else {
    set interact "-background"
  }
  set xservname\
                "[string tolower $TeXmodeVars(nameOfTeXProgram)]"
  if {!$bg && $showTeXLog == 2} {
    ::xserv::addEndExecHook $xservname TeX::showLogHook
  } else {
    ::xserv::removeEndExecHook $xservname TeX::showLogHook
  }

  set status [::xserv::invoke $interact $xservname \
           -options "$TeXmodeVars(additionalTeXFlags)"  -format\
                     "[TeX::effectiveFormat]"  -file $filename]
  status::msg "$status"
}

#
# End-exec hook used to open the log file when requested.
#
proc TeX::showLogHook {implArray argsArray result} {
  array set impl $implArray
  array set args $argsArray
  if {!$args(xservInteraction) || ($impl(mode) == "App")} {
    return
  }
  set filename $args(file)
  set wins [winNames -f]
  set logname "[file rootname $filename].log"
  set idx [lsearch -exact $wins $logname]
  if {$idx == -1} {
    edit -r -w "[file rootname $filename].log"
  } else {
    bringToFront "$logname"
    revert
  }
}

###
## For diff mode
###
::xserv::declare Diff "Show differences between files" \
                                      oldfile newfile {options ""}

::xserv::register Diff GNUdiff  -sig DIFF -path\
                    [file join $HOME Tools "GNU Diff"]  -driver {
    array set diffopts $params(options)
    set flags $diffopts(diffFlags)
    if {$diffopts(linesOfContext) != 0} {
        lappend flags -C $diffopts(linesOfContext)
    }
    if {$diffopts(treatAllFilesAsText)} {
        lappend flags -a
    }
```

```
    if {$diffopts(ignoreCase)} {
        lappend flags -i
    }
    if {$diffopts(ignoreBlankLines)} {
        lappend flags -B
    }
    if {$diffopts(ignoreSpaceChanges)} {
        lappend flags -b
    }
    if {$diffopts(ignoreWhiteSpace)} {
        lappend flags -w
    }
    if {$diffopts(compareDirectoriesRecursively)} {
        lappend flags -r
    }
    tclAE::build::resultData -n $params(xservTarget) misc\
         dosc  --- [tclAE::build::TEXT "$flags $params(oldfile)\
                                              $params(newfile)"]
}

::xserv::register Diff DiffBOA -sig DifB -driver {
    set aevt [list tclAE::build::resultData -n\
                                  $params(xservTarget) Diff Diff]
    lappend aevt Oldf [tclAE::build::alis $params(oldfile)]
    lappend aevt Newf [tclAE::build::alis $params(newfile)]
    array set diffopts $params(options)
    if {$diffopts(linesOfContext) != 0} {
      lappend aevt Frmt 2 Cntx $diffopts(linesOfContext)
    }
    regsub { <[0-9]+>} [winNames -f] "" windows
    set idx [lsearch $windows [quote::Glob $params(oldfile)]]
    if {$idx >= 0} {
      getWinInfo -w [lindex [winNames -f] $idx] winfo
      switch -exact -- $winfo(platform) {
        "mac"   { lappend aevt Eol1 1 }
        "unix"  { lappend aevt Eol1 2 }
        "win"   { lappend aevt Eol1 3 }
      }
    }
    set idx [lsearch $windows [quote::Glob $params(newfile)]]
    if {$idx >= 0} {
      getWinInfo -w [lindex [winNames -f] $idx] winfo
      switch -exact -- $winfo(platform) {
        "mac"   { lappend aevt Eol2 1 }
        "unix"  { lappend aevt Eol2 2 }
        "win"   { lappend aevt Eol2 3 }
      }
    }
    eval $aevt
  }

::xserv::register Diff diff -driver {
  set cmdline [list $params(xserv-diff)]
  array set diffopts $params(options)
```

```
  foreach opt $diffopts(diffFlags) {
    lappend cmdline $opt
  }
  if {$diffopts(linesOfContext) != 0} {
      lappend cmdline -C $diffopts(linesOfContext)
  }
  if {$diffopts(treatAllFilesAsText)} {
      lappend cmdline -a
  }
  if {$diffopts(ignoreCase)} {
      lappend cmdline -i
  }
  if {$diffopts(ignoreBlankLines)} {
      lappend cmdline -B
  }
  if {$diffopts(ignoreSpaceChanges)} {
      lappend cmdline -b
  }
  if {$diffopts(ignoreWhiteSpace)} {
      lappend cmdline -w
  }
  if {$diffopts(compareDirectoriesRecursively)} {
      lappend cmdline -r
  }
  lappend cmdline $params(oldfile) $params(newfile)
  return $cmdline
} -progs {diff}

proc Diff::execute\
       {{isdir 0} {name {* File Comparison *}} {storeResult 0}} {
  global DiffmodeVars Diff::1 Diff::2 win::Modes HOME  diffDir\
            Diff::result Diff::1Open Diff::2Open  Diff::leftDir\
                              Diff::rightDir DiffSig tcl_platform

  set Diff::leftDir ""
  set Diff::rightDir ""
  set diffDir $isdir

  #status::msg "Launching 'GNU Diff'"
  set flags $DiffmodeVars(diffFlags)
  status::msg "Starting diff..."
  # The MacOS diff is a bit peculiar with funny filenames.
  if {$tcl_platform(platform) == "macintosh"} {
    set oldfile "\"[win::StripCount ${Diff::1}]\""
    set newfile "\"[win::StripCount ${Diff::2}]\""
  } else {
    set oldfile "[win::StripCount ${Diff::1}]"
    set newfile "[win::StripCount ${Diff::2}]"
  }
  set dtext\
         [ ::xserv::invoke -foreground Diff  -oldfile $oldfile \
        -newfile $newfile  -options [array get DiffmodeVars]  ]
  status::msg "Starting diff ... done"

```

```
  if {[lsearch -exact [winNames -f] ${Diff::1}] >= 0} {
    set Diff::1Open 1
  } else {
    set Diff::1Open 0
  }
  if {[lsearch -exact [winNames -f] ${Diff::2}] >= 0} {
    set Diff::2Open 1
  } else {
    set Diff::2Open 0
  }

  if {![string length $dtext] || (!$diffDir &&\
            [regexp {^Files.*are identical[\r\n]*$} $dtext])} {
    if {!$storeResult} {
      alertnote "No difference:\r${Diff::1}\r${Diff::2}"
    }
    return 0
  } else {
    # If requested, return the diff result in Diff::result,
    # rather than opening a diff window
    if {$storeResult} {
      set Diff::result $dtext
    } else {
      Diff::diffWindow $dtext $name
    }
    return 1
  }
}

###
## For HTML mode
###
::xserv::declare viewURL {Display the target of an URL} url

::xserv::register viewURL {Internet Explorer} -sig MSIE\
                                                    -driver {
  tclAE::send $params(xservTarget) WWW! OURL  ----\
                                [tclAE::build::TEXT $params(url)]
}

::xserv::register viewURL {Apple help browser} -sig hbwr\
                                                    -driver {
  set urltype [lindex [split $params(url) ":"] 0]
  if {$urltype != "file"} {
    alertnote "Apple help browser can only display files"
    return
  }
  tclAE::send $params(xservTarget) aevt odoc  ----\
         [tclAE::build::alis [string range $params(url) 7 end]]
}

::xserv::register viewURL {Safari} -sig sfri -driver {
    tclAE::send $params(xservTarget) GURL GURL  ----\
                                [tclAE::build::TEXT $params(url)]
```

```
}

proc htmlSendWindow {{path ""} {url 0}} {
  global HTMLmodeVars browserSig tcl_platform alpha::macos

  if {$path == ""} {
    if {![llength [winNames]]} {return}
    set path [html::StrippedFrontWindowPath]

    if {[winDirty]} {
      html::SaveBeforeSending $path
      # Get path again, in case it was Untitled before.
      set path [html::StrippedFrontWindowPath]
      if {![file exists $path]} {
        alertnote "Can't send window to browser."
        return
      }
    }
  }
  if {$url} {
    set path [join [lrange [html::BASEfromPath $path] 0 2] ""]
  } else {
    set path [quote::Url $path\
                  [expr {$tcl_platform(platform) == "macintosh"}]]
    if {${alpha::macos}} {
      regsub -all : $path / path
      if {$tcl_platform(platform) == "macintosh"}\
                                              {set path "/$path"}
    }
    set path "file://$path"
  }

  if {$HTMLmodeVars(browseInForeground)} {
    xserv::invoke -foreground viewURL -url "$path"
  } else {
    xserv::invoke -background viewURL -url "$path"
  }
}

⟨/examples⟩
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition. By tradition there should also be a lot of numbers in roman that refer to the code lines where the entry is used, but unfortunately I don't have a convenient way of cross-referencing Tcl code to generate that information.