Accessing and Administering Applications in CDE

Setting up transparent access to applications and resources in a highly networked environment is made simpler by facilities that enable system administrators to integrate applications into the CDE desktop.

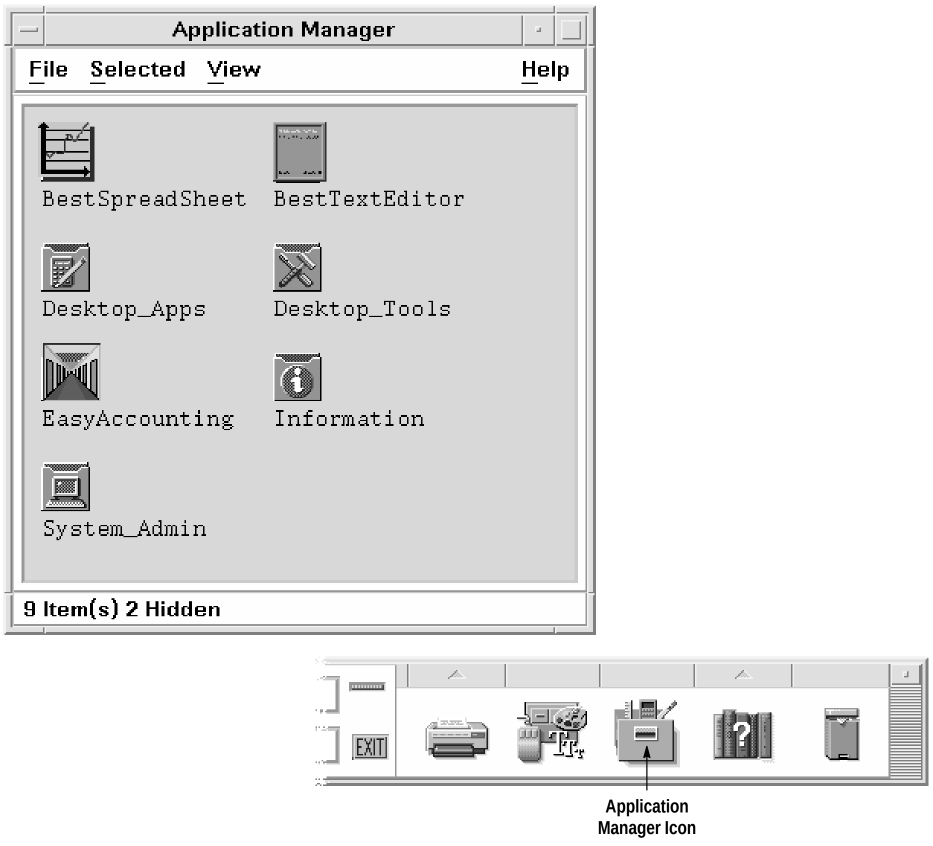
by Anna Ellendman and William R. Yoder

A major purpose of a graphical desktop is to make it easier for users to browse and run applications and to identify and manipulate the applications' data files. Since UNIX® systems are generally highly networked, it is desirable that a desktop also provide network transparency—the ability to launch applications and supply data to them without worrying about where in the network the applications and data are located.

Wherever possible, these ease-of-use features should not be provided at the expense of system administrators. There should be a standard procedure for incorporating preexisting applications into the desktop, and the desktop should provide tools for performing these procedures.

This article describes how users locate and launch applications from the CDE desktop and how system administrators integrate applications into the desktop's graphical environment. The information is also relevant for application developers, since the administration model and tools for integrating existing applications are also used to provide basic desktop integration for new applications.

Fig. 1. *CDE application manager window and the front panel icon for opening the window.*



User Model for Applications

CDE makes it easier for users to access and run applications by providing:

- A way to represent an application as an icon. The user can start an application by double-clicking the icon. These icons are called application icons.
- A special container for application icons. This container is called the *application manager*.
- A way to specify the unique appearance and behavior for icons representing an application's data files.

Application Manager and Application Icons. The application manager is a single container for all the applications available to the user and is opened by clicking a control in the CDE front panel (see Fig. 1). Each item in the top level of the application manager is a directory containing one or more related applications. These directories are called *application groups*.

By convention, an application group contains, for each application in the group, the application icon that starts the application, plus other files related to the application such as sample data files, templates, and online information (see Fig. 2). The system administrator can organize the application groups into deeper hierarchies.

Since the application groups are displayed together in a single window, they appear to occupy the same file system location. This makes applications easy to find and launch. The fact that this is not the case, and that the application groups are actually located in a variety of local and remote locations, is hidden from users.

From the end user's point of view, the application manager window is owned by the system. The user does not have the ability to create or move icons in the window directly.

Data Files and File Manager. Like the application manager, the *file manager* represents objects as icons. Frequently, these objects are the application data files. The desktop provides a way to specify the behavior of various types of data files. This makes it possible for users to start an application from the file manager by double-clicking one of its data files, by dropping a data file on the application icon, or by choosing Open from the data file's pop-up menu (see Fig. 3).

Application Manager Administration

The application manager is designed to meet several important requirements:

- It must appear to be a single location into which applications are gathered from a variety of locations.
- It must be customizable on a personal (per-user) or system-wide (per-workstation) basis.
- It must be network capable, that is, it must be able to gather applications located on other systems.
- It must be dynamic so that its contents can be changed when applications are added to or removed from the local system or application servers in the network.

To meet these requirements, the CDE designers chose to make the application manager a file manager view of a special temporary directory. The application manager directory is created automatically when the user logs in at the location:

```
/var/dt/appconfig/appmanager/login-display
```

Fig. 2. *Contents of an application group for a single application.*

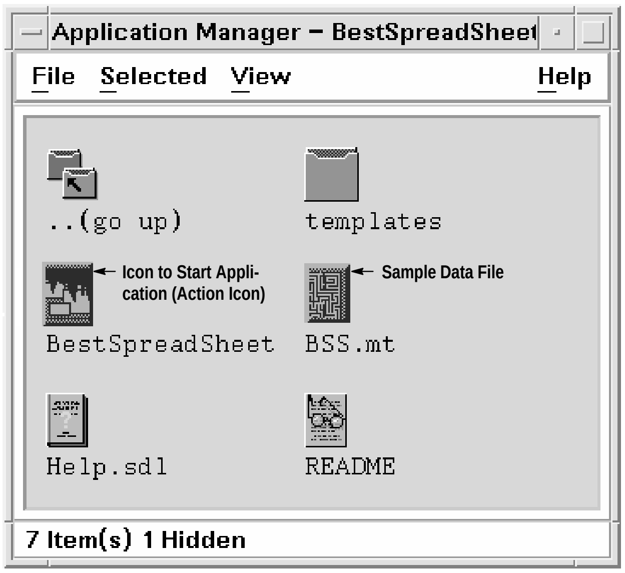
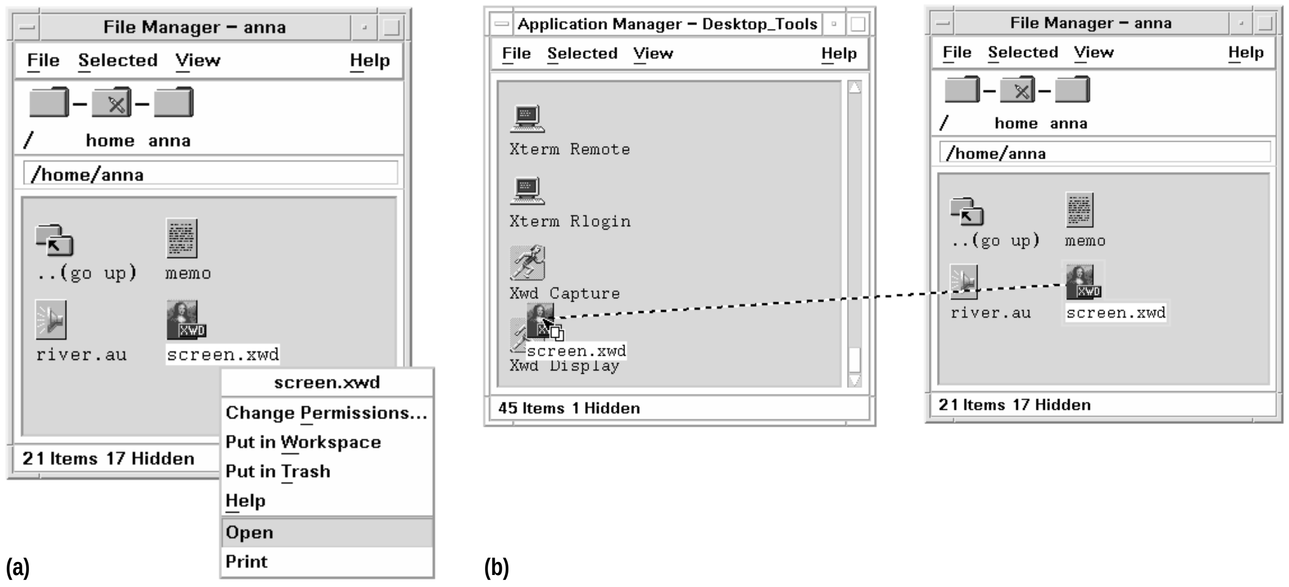


Fig. 3. Running an application using (a) a data file's pop-up menu in the file manager or (b) drag and drop between the file manager and the application manager.



For example, when user anna logs into CDE at display hpcvxpae:0, the CDE login manager creates the directory:

```
/var/dt/appconfig/appmanager/anna-hpcvxpae-0
```

The directory is both user and display dependent. Each user gets an application manager directory, and a user logging into more than one system obtains a separate application manager directory on each system. This is necessary to allow the application manager to be customized on both a per-user and a per-system basis.

The application manager directory persists for the length of the user's CDE session and is recreated each time the user logs in. The temporary nature of the application manager is possible because none of its contents are actually located in its file system location.

Gathering Applications. After the login manager creates the application manager directory, it must gather application groups into the directory. The application groups are gathered by means of symbolic links, and the links are made from multiple locations. This is what makes it possible for the application manager to display application groups located in a variety of directories, including personal, system-wide, and remote locations.

The symbolic links that gather the application groups are created by the CDE utility dtappgather, which is automatically run by the login manager each time the user logs in. For example, the desktop provides a built-in application group:

```
/usr/dt/appconfig/appmanager/C/Desktop_Apps
```

At login time, dtappgather creates the following symbolic link:

```
/var/dt/appconfig/appmanager/anna-hpcvxpae-0/
 Desktop_Apps->
/usr/dt/appconfig/appmanager/C/Desktop_Apps
```

The result is that the application manager contains the Desktop_Apps application group (see Fig. 4).

Application Search Path. To gather application groups from various locations, dtappgather requires a list of locations containing the application groups to be gathered. This list of locations is called the desktop's application search path.

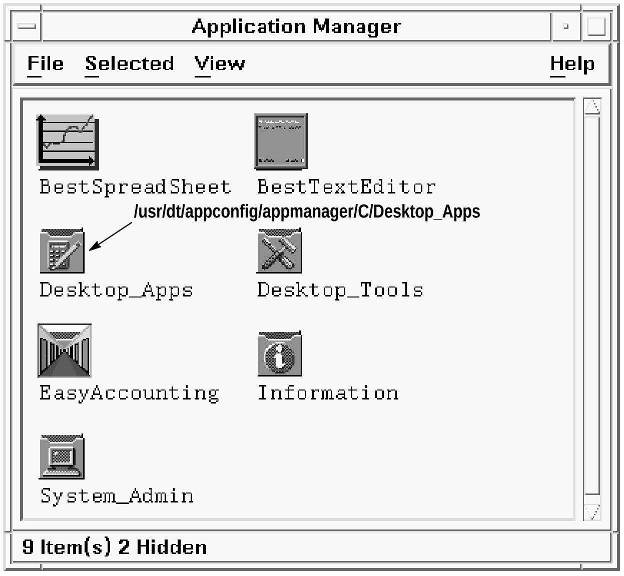
The default application search path consists of three local locations:

Personal	**$HOME/.dt/appconfig/appmanager**
System-wide	**/etc/dt/appconfig/appmanager/<$LANG>**
Built-in	**/usr/dt/appconfig/appmanager/<$LANG>**

The built-in location is used for factory-installed application groups. System administrators can add application groups to the system-wide location to make those applications available to all users logging into the system. The personal location allows a user to add application groups that are available only to that single user.

System administrators or users may need to add other locations to the application search path. CDE provides two environment variables that modify the application search path: the system-wide variable DTSPSYSAPPHOSTS and the personal variable DTSPUSERAPPHOSTS.

Fig. 4. The Desktop_Apps *application group is a built-in group provided by the desktop.*



The entire application search path, consisting of the default locations plus the additional locations specified by the environment variables, is created by a CDE utility named dtsearchpath. The login manager runs the dtsearchpath utility just before it runs dtappgather. The dtsearchpath utility uses a set of rules to define how the total value of the search path is assembled. Directories listed in the personal environment variable have precedence over the default personal location, and personal locations have precedence over system-wide locations.

The most common reason for modifying the application search path is to add remote locations on application servers so that those remote applications can be easily started by users. The application search path variables accept a special syntax that makes it easy to add application servers. For example, VARIABLE=hostname: is expanded by dtappgather (assuming NFS mounts) to the system-wide location on hostname:

```
/net/<hostname>/etc/dt/appconfig/appmanager/ <$LANG>
```

For example, if DTSPSYSAPPHOSTS specifies two remote systems:

```
DTSPSYSAPPHOSTS=SystemA:,SystemB:
```

and these systems contain the following application groups:

```
SystemA  /etc/dt/appconfig/appmanager/C/
          EasyAccounting
SystemB  /etc/dt/appconfig/appmanager/C/
         BestSpreadSheet
```

then dtappgather creates the following symbolic links:

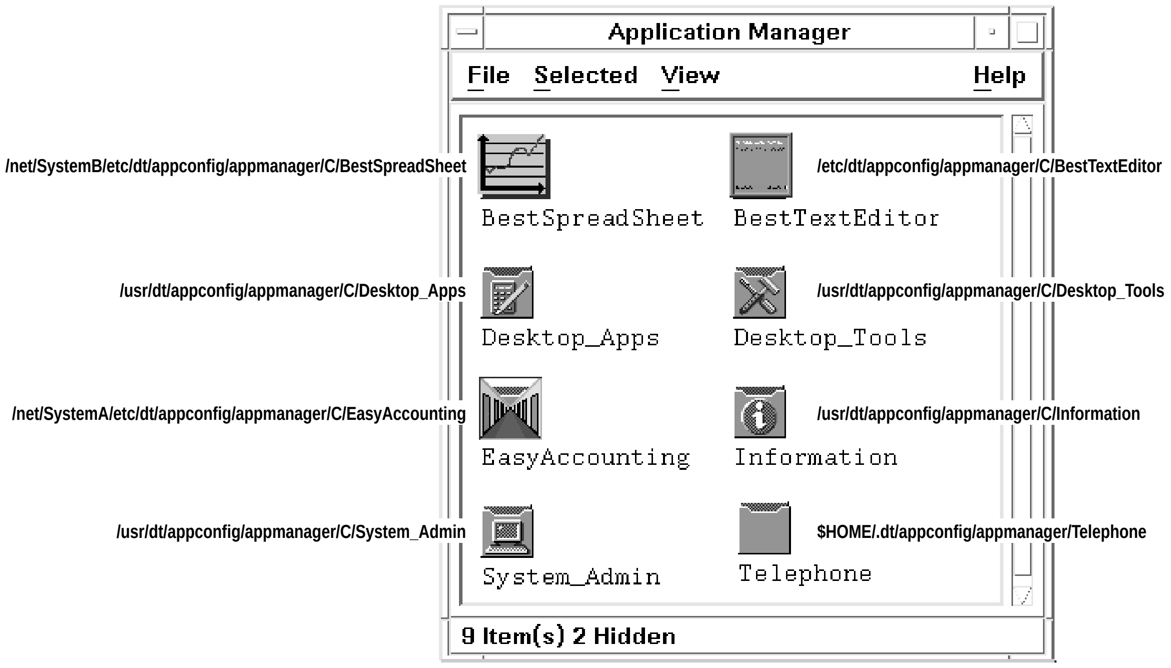
```
/var/dt/appconfig/appmanager/anna-
hpcvxpae-0/EasyAccounting ->
/net/SystemA/etc/dt/appconfig/appmanager/C/
   EasyAccounting

/var/dt/appconfig/appmanager/anna–
hpcvxpae–0/BestSpreadSheet ->
/net/SystemB/etc/dt/appconfig/appmanager/C/
   BestSpreadSheet.
```

If the system uses a mount point other than /net, the system administrator can set a desktop environment variable DTMOUNTPOINT to specify the mount point location.

Fig. 5 shows an application manager containing application groups gathered from personal, system-wide, and built-in locations and from two application servers.

Fig. 5. *The application manager gathers application groups on the application search path.*


Application Manager
File
Selected
View
Help
/net/SystemB/etc/dt/appconfig/appmanager/C/BestSpreadSheet
BestSpreadSheet
/etc/dt/appconfig/appmanager/C/BestTextEditor
BestTextEditor
/usr/dt/appconfig/appmanager/C/Desktop_Apps
Desktop_Apps
/usr/dt/appconfig/appmanager/C/Desktop_Tools
Desktop_Tools
/net/SystemA/etc/dt/appconfig/appmanager/C/EasyAccounting
EasyAccounting
/usr/dt/appconfig/appmanager/C/Information
Information
/usr/dt/appconfig/appmanager/C/System_Admin
System_Admin
$HOME/.dt/appconfig/appmanager/Telephone
Telephone
9 Item(s) 2 Hidden

Application Infrastructure

The ability to represent applications and data files as icons that have meaningful behavior and appearance on the desktop is made possible by an infrastructure of desktop constructs and configuration files. These constructs are actions, data types, and icon image files.

Actions. The desktop uses actions to create a relationship between an icon in the application manager (or file manager) and a command. Actions are the constructs that allow you to create application icons (icons that the user can double-click to run applications).

For example, consider the following definition for an action named Xwud:

```
ACTION Xwud
{
  LABEL      Xwd_Display
  ICON       XwudIcon
  ARG_TYPE   XWD
  WINDOW_TYPENO_STDIO
  DESCRIPTIONDisplays an X Windows screen\
             file
  EXEC_STRING  xwud –in %Arg_1”XWD file:”%
}
```

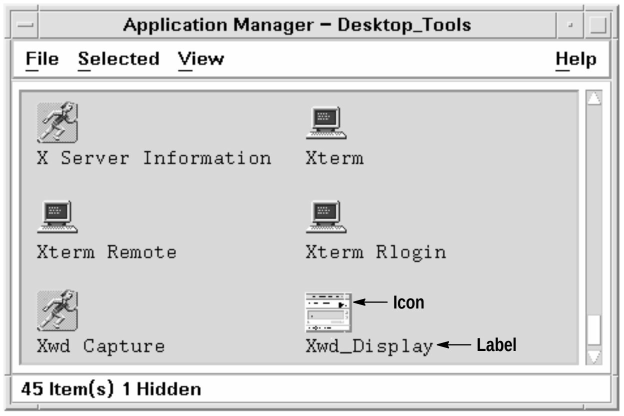
The desktop assembles and maintains a database of action definitions, including built-in actions and others created by users and system administrators. Once an action is defined in the actions database, it can be visually represented in the application manager or file manager as an icon. This icon is called an application icon because the underlying action usually is written to launch an application. Since icons in the file manager and the application manager represent files, creating an application icon involves creating a file that is related to the action. The relationship is provided by giving the file the same name as the action (in this case, Xwud), and by making the file executable. The real content of the file is irrelevant. Fig. 6 shows an application icon for the Xwud action.

The ICON and LABEL fields in the Xwud action definition describe the appearance—the icon image used and the text label for that icon. The DESCRIPTION and EXEC_STRING fields describe the icon's behavior. The DESCRIPTION field contains the text displayed in the CDE help viewer when the user selects the icon and requests help (**F1**). The EXEC_STRING specifies the command that is run when the user double-clicks the icon. For the Xwud action, the command is:

```
xwud –in <file>
```

where file is the file argument.

Fig. 6. *Icon for the Xwud action.*



The EXEC_STRING field uses a special syntax to represent file arguments. For example, the file argument in the Xwud action is represented as:

```
%Arg_1"XWD file:"%
```

This syntax specifies how the application icon treats data files. If the user drops a data file on the icon labeled Xwd_Display, its path name, supplied by the desktop drag and drop infrastructure, is used as the argument. If the user double-clicks the icon, the action prompts for that path by displaying a dialog box with a text field labeled XWD file:, which indicates that user must enter the file name of an X Window dump (.xwd) file.

The ARG_TYPE field limits the action to a particular type of data. If the user tries to use a different type of file, either by dropping the file on the action icon or responding to the prompt, the action returns an error dialog box.

Data Types. Files and directories are represented in the CDE file manager as icons that users can manipulate. There is little advantage to this iconic representation unless the desktop can supply meaningful appearance and behavior to these icons. Data typing provides this capability.

For example, directories appear in the file manager as folder-like icons, and users open a view of a directory by double-clicking its icon. That behavior, which is unique to directories, is provided by data typing.

The most common use for data typing is to provide a connection between data files and their applications. If an application reads and writes a special type of data, it is useful to create a data type for the data files. This data type might specify that the data files use a special icon image and that double-clicking a data file runs the application and loads the data file.

A data type definition has two parts:

- DATA_CRITERIA: specifies the criteria used to assign a file to that data type.
- DATA_ATTRIBUTES: defines a file's appearance and behavior.

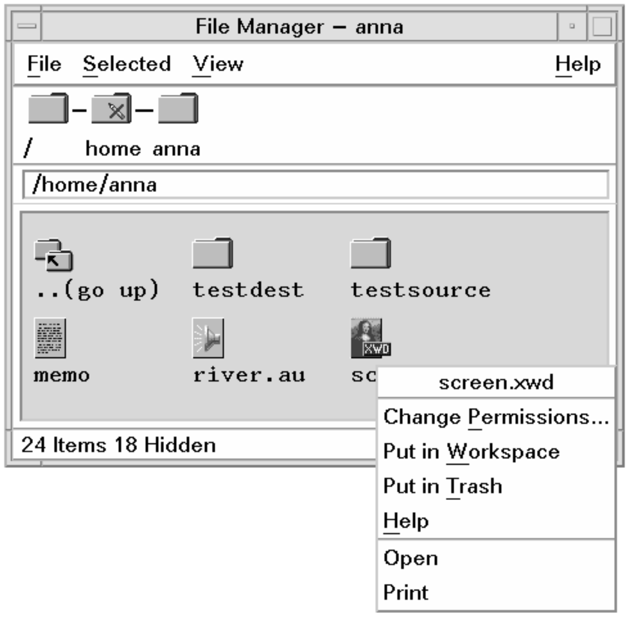
For example, here is a data type definition for X Window dump files (the data files for the Xwud action):

```
DATA_CRITERIA XWD1
{
  DATA_ATTRIBUTES_NAME XWD
  MODE              f
  NAME_PATTERN        *.xwd
}

DATA_ATTRIBUTES XWD
{
ACTIONS    Open,Print
ICON     Dtxwd
DESCRIPTIONThis file contains  \
           an XWD graphic image.
}
```

The criteria definition specifies that the data type applies to files (MODE f) whose names (NAME_PATTERN) end with the suffix .xwd. (CDE also supports content-based data typing.)

Fig. 7. *An .xwd data file in the file manager, with Open and Print actions in its pop-up menu.*



The ACTIONS field in the attributes definition describes the file's behavior. The first entry (in this case, Open) describes the file's double-click behavior. All the entries in the ACTIONS list also appear in the file manager Selected menu (see Fig. 7).

In the desktop, Open and Print are general action names that are used with many data types. For .xwd files, the Open action is a synonym for the Xwud action, and the desktop must provide a connection between them. This connection is made through another action definition in which an action named Open is mapped to the Xwud action for the .xwd data type:

```
ACTION Open
{
TYPE       MAP
MAP_ACTION Xwud
DATA_TYPE    XWD
}
```

When a user selects a data file in the file manager and runs the Open action on it (by double-clicking the file or by choosing Open from the Selected menu), the desktop searches for the appropriate Open action for that data type. Since this action is mapped to the Xwud action, the desktop then runs the Xwud action, and passes the selected data file to it as the file argument.

The Print action is handled similarly to Open. The following declaration is a set of actions for printing .xwd files:

```
ACTION Print
{
  TYPE       MAP
  MAP_ACTION XWD_Print
  DATA_TYPE    XWD
}

ACTION XWD_Print
{
  ARG_TYPE   XWD
  EXEC_STRING  /bin/sh –c 'cat %(File)Arg_1%\
              |xwd2sb|pcltrans –s –R –e2 >  \
              $HOME/temp.pcl;        \
              dtaction PrintRaw $HOME/temp.pcl;
\
              rm $HOME/temp.pcl'
{
```

The XWD_Print action illustrates two additional features of actions:

- An action can invoke a shell (bin/sh in this case)

- An action can invoke another action. This is done using the dtaction command. The XWD_Print command changes the .xwd file to a PCL file and then invokes a built-in action named PrintRaw which prints PCL files.

Article 3 describes the different types of data structures and databases associated with CDE action and data types.

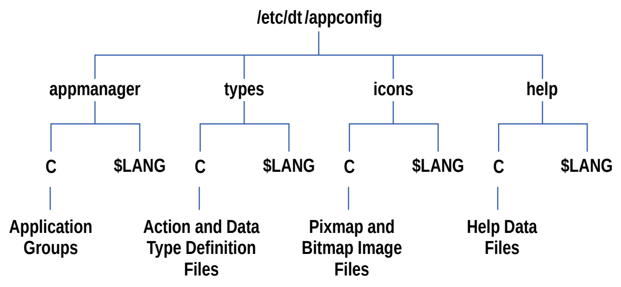
Icon Image Files. Since the desktop makes heavy use of icons to represent files, directories, and applications, icon image files are an important part of the suite of desktop configuration files.

A particular object on the desktop generally requires several icons. For example, the file manager has viewing preferences for representing files as large icons (32 by 32 pixels) or small icons (16 by 16 pixels). Furthermore, the desktop uses pixmaps for color displays and bitmaps for monochrome displays. To differentiate icons by size and type (pixmap or bitmap), icon files use the naming convention basename.size.type. For example, a large pixmap for the Xwud action might be named Xwud.l.pm.

The icon image used by an object is specified in its action or data type definitions by the ICON field (see examples above), which specifies the icon image to use in the file manager and the application manager. The ICON field specifies only the base name, and the desktop adds the appropriate file name extensions, depending on the file manager view setting and type of display. Furthermore, the ICON field does not specify a path. The desktop uses search paths to locate icons and other parts of the desktop infrastructure.

Locating Actions and Icons. As with application groups, the desktop uses search paths to locate action and data type definitions and icon files. Each application search path location has a set of corresponding locations for actions, data types, and icons. For example, Fig. 8 shows the default system-wide search path locations. The types directory for actions and data types and the icons directory for icon image files are analogous to the appmanager directory for the application groups.

Fig. 8. *Directory structure of system-wide desktop configuration files.*



The help directory is used for application help files created using the CDE Help Developer's Kit. CDE help is described in ***Article 5***.

The search paths for action, data type, icon, and help files are automatically updated when the application search path is modified. This ensures that the desktop will find all the desktop configuration files needed by the application.

For example, if SystemA: is added to the application search path, the locations:

```
/net/SystemA/etc/dt/appconfig/types/<$LANG>
/net/SystemA/etc/dt/appconfig/icons/<$LANG>
/net/SystemA/etc/dt/appconfig/help/<$LANG>
```

are automatically added to the action/data type, icon, and help search paths.

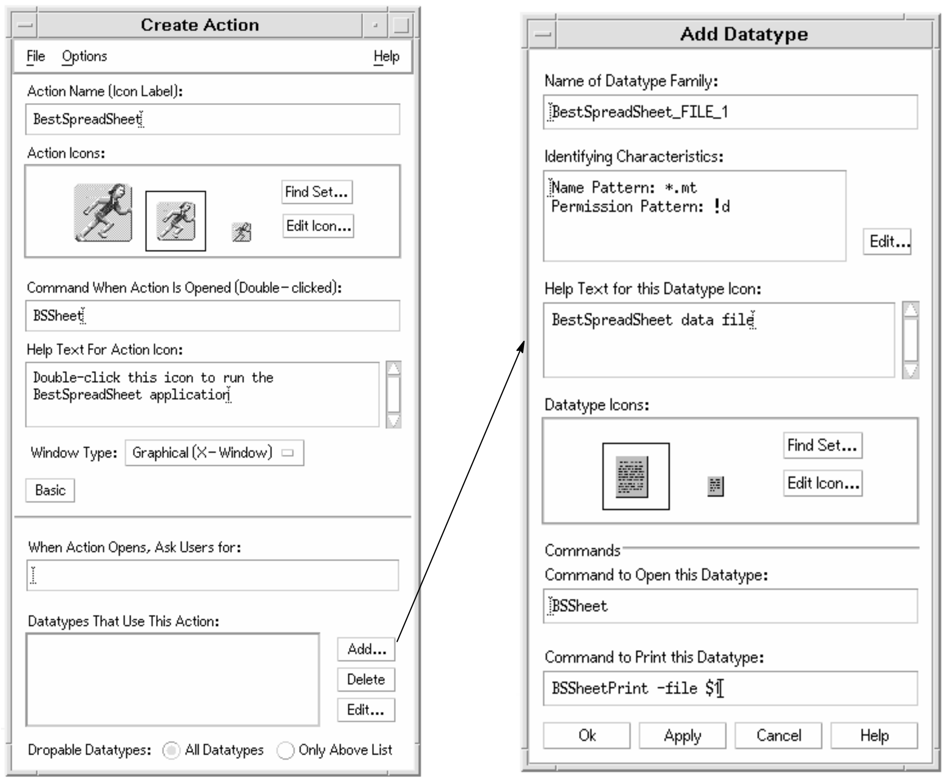
Create Action

The syntax of action and data-type definitions provides a great deal of flexibility in specifying desktop behavior. While this makes the definitions very powerful, it also makes the syntax somewhat complex.

The desktop includes an application, *create action*, that allows users and system administrators to create typical actions and data types without having to learn the syntax rules for the definitions. Create action provides fill-in-the-blank text fields for supplying various parts of the action and data type definitions and provides a way to browse and choose from available icons (see Fig. 9). Furthermore, create action allows the user to enter the command to be executed (EXEC_STRING) using shell language for file arguments (e.g., $n rather than %(File)Arg_n%).

Create action is optimized for creating actions and data types for applications. It creates an action to launch the application and one or more data types for the application. For each data type, create action produces an Open command that runs the application. Optionally, it can also create a Print action.

Fig. 9. *Create action main window and dialog box.*



Create Action

File Options Help

Action Name (Icon Label):

BestSpreadSheet

Action Icons:

Find Set... Edit Icon...

Command When Action Is Opened (Double-clicked):

BSSheet

Help Text For Action Icon:

```
Double-click this icon to run the
BestSpreadSheet application
```

Window Type: Graphical (X-Window)

Basic

When Action Opens, Ask Users for:

Datatypes That Use This Action:

Add... Delete Edit...

Dropable Datatypes: All Datatypes Only Above List

Add Datatype

Name of Datatype Family:

BestSpreadSheet_FILE_1

Identifying Characteristics:

```
Name Pattern: *.mt
Permission Pattern: !d
```

Edit...

Help Text for this Datatype Icon:

```
BestSpreadSheet data file
```

Datatype Icons:

Find Set... Edit Icon...

Commands

Command to Open this Datatype:

BSSheet

Command to Print this Datatype:

```
BSSheetPrint -file $1
```

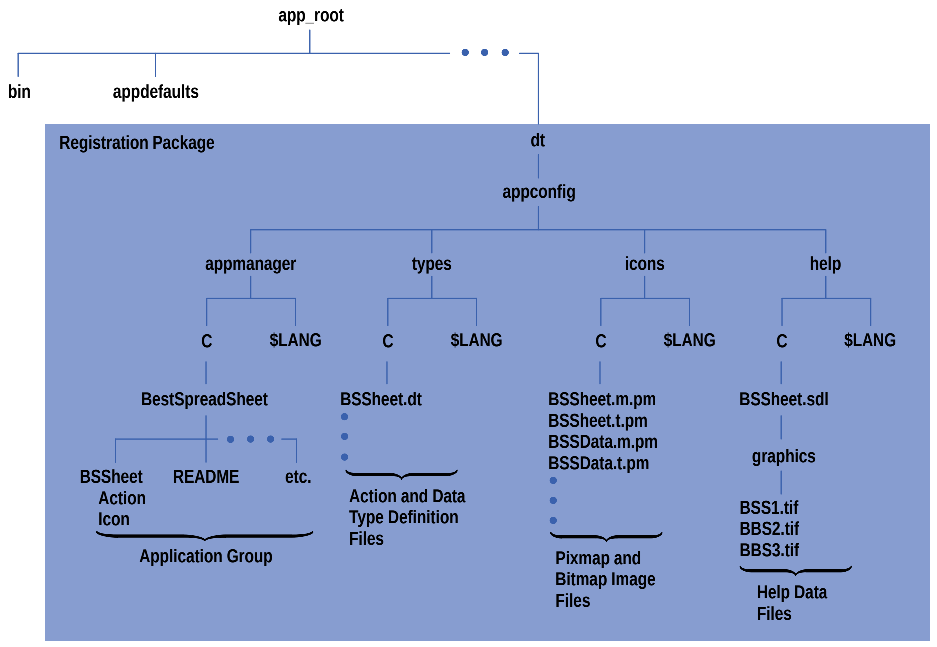
Ok Apply Cancel Help

Application Integration

An application can be integrated into the desktop by placing its desktop configuration files into the locations specified by the desktop search paths shown in Fig. 8. However, this can make administration difficult because the files are scattered among numerous directories, each of which contains files for many applications.

It is usually preferable to gather all the configuration files for an application in one location, called the application root (app_root). Applications generally are installed this way. For example, the files for an audio application might be

Fig. 10. *Example of a registration package.*



installed under /opt/audio. However, since the directories under the app_root are not on the desktop search paths, the application groups, actions, data types, and icons would not be found unless the search paths were modified.

Application Registration. CDE provides the utility dtappintegrate which lets system administrators install desktop-related application files under an app_root directory without modifying the desktop search paths. The function of dtappintegrate is to create links between the desktop files in the app_root location and the system-wide search path locations. The process of creating these links with dtappintegrate is called application registration.

The collection of desktop configuration files beneath the app_root directory is called the registration package. Fig. 10 illustrates a registration package for an application named BestSpreadSheet.

The registration package contains the desktop configuration files needed by the application, including the application group, actions, data types, icons, and help files. The registration package uses the same directory organization as the search paths (compare Fig. 10 with Fig. 8).

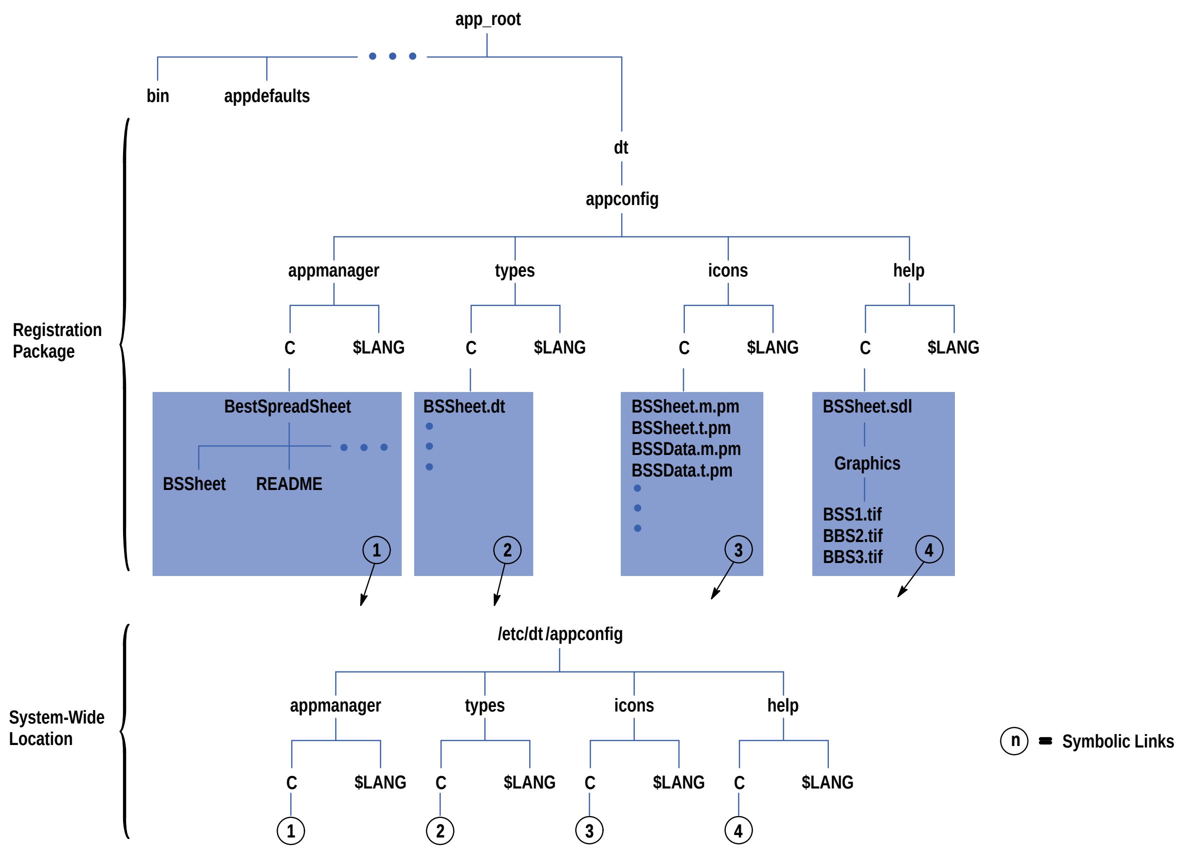
Once the registration package has been created, the registration is accomplished by running the dtappintegrate utility, which creates the symbolic links shown in Fig. 11.

The system administrator can also use dtappintegrate to break the symbolic links for an application's desktop configuration files. This is necessary to move an application to a different application server. In addition, the system administrator can also use dtappintegrate to move the application configuration registry to a different location on the application server to control which users can access an application from the desktop.

See the sidebar: ***Application Servers and Clients in CDE***.

Regathering Applications. Applications registered by dtappintegrate do not become available from the application manager until the utility that gathers applications from search path locations, dtappgather, is rerun. Since dtappgather runs automatically at login time, newly registered applications become available when the user logs out and back in again. To save users the inconvenience of logging out, the desktop provides an action named Reload Applications. The user can run this action by opening the application manager and double-clicking the Reload Applications icon.

Fig. 11. *Links created by dtappintegrate.*



Conclusion

CDE provides the ability to represent applications and their data files as icons, and supplies a single container for applications called the application manager. Applications are gathered into the application manager from multiple locations specified by the application search path. Remote locations can be added to the application search path so that applications on networked application servers can be integrated seamlessly into the application manager. The infrastructure required to represent applications as icons consists of actions, data types, and icon image files. These files can be placed into a registration package for the application and then registered onto the desktop using the dtappintegrate utility.

Acknowledgments

Thanks to Jay Lundell, Bob Miller, and Brian Cripe for contributing to the design of the application administration model for CDE and to Julia Blackmore for her great attention to detail in reviewing the documentation. Special thanks to John Bertani for his work on the search path utilities and to Ron Voll for his work on actions. Finally, we were fortunate to have the opportunity to work with Troy Cline and Sandy Amin at IBM as they implemented dtappintegrate and the file manager and with Rich McAllister of SunSoft who contributed the CDE file mapping routines.

▶ Go to Article 3

▶ Go to Table of Contents

▶ Go to HP Journal Home Page