The CDE Action and Data Typing Services

Several different types of databases and their associated APIs are involved in determining the look and behavior of icons presented in the Common Desktop Environment.

by Arthur F. Barstow

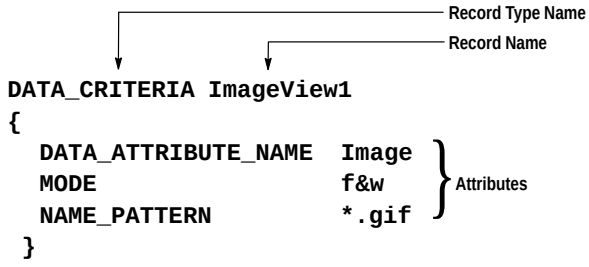
Two fundamental requirements of a computerized desktop system are a unified view of a user's data and a consistent access method to the data. This article describes how Hewlett-Packard's Common Desktop Environment (CDE) meets these requirements through the use of its data typing and action services. The data typing service defines attributes of a data type such as its appearance (icon) and behavior (action). The action service provides mechanisms for linking a data type's behavior to its associated application or execution command.

The data typing service and the action service both use databases. The data typing service database contains data criteria and data attribute records, and the action service database contains action records. The term database in the context of data typing and action databases refers to records stored in flat, ASCII files and not in a generalized database service.

Applications wanting to use these CDE services must first load the databases into the application's memory. Once loaded, application programming interfaces (APIs) described in this article can be used to access the database. CDE only provides APIs to retrieve data from the databases.

Each database record consists of the record type name, the name of the record, and one or more attributes (also called fields). Each attribute has a name and a value and each attribute must begin on a separate line. Fig. 1 shows the basic elements of a database record.

Fig. 1. *The basic structure of records contained in the CDE data typing and action databases.*



Although CDE defines numerous data types and actions in its default databases, these databases can be augmented with system-wide and user-defined records. User-defined definitions have the highest precedence, followed by systemwide definitions. The default databases have the lowest precedence.

Fig. 2 shows an overview of the CDE data typing and action services and and their interrelationships. These data structures and their relationships are described in this article.

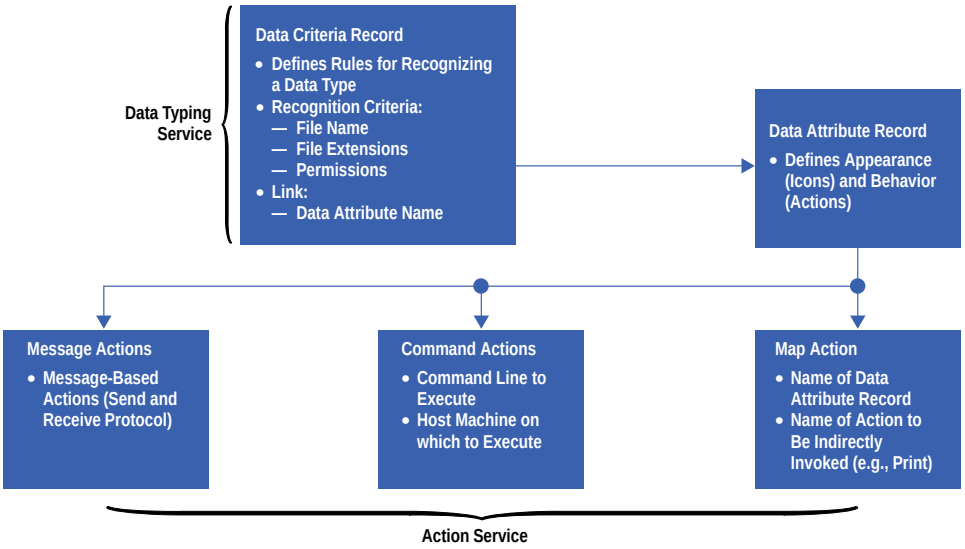
The Data Typing Service

As mentioned above, the data typing service contains data criteria and data attribute records. Data criteria records contain rules for recognizing a data type. The rules may specify a data type's file name extension or file permissions. Data attribute records are used to define a data type's appearance and behavior such as a type's icon and associated applications. A data type can be defined for a file or a buffer of memory.

Data Criteria Records. Data criteria records define the rules for recognizing or identifying a data type by using the following criteria:

- File name (including shell pattern matching symbols)
- Contents (may be a file or memory buffer)
- File permissions and file type (a file type may be a directory, symbolic link, etc.).

Fig. 2. *An overview of the relationships between data typing and action service data strucures.*



The following declaration contains three data criteria records for an image viewer application that can display files and data in formats such as X11 bitmaps, GIF images, and X11 pixmaps.

```
DATA_CRITERIA ImageView1
{
  DATA_ATTRIBUTE_NAME Image
  MODE              f&w
  NAME_PATTERN        *.gif
}

DATA_CRITERIA ImageView2
{
  DATA_ATTRIBUTE_NAME Image
  PATH_PATTERN        */bitmaps/*.bm
}

DATA_CRITERIA ImageView3
{
  DATA_ATTRIBUTE_NAME Image
  CONTENT             0 string #define
}
```

All of these records refer to the data type Image. The ImageView1 record will match any writable file with a file name extension of .gif. The ImageView2 record will match any file ending in .bm, but the file must be in a directory named bitmaps. The ImageView3 record will match any file containing the string #define at the beginning of the file. Fig. 3 contains a pseudo-BNF (Backus-Nauer form) for a data criteria record.

Fig. 3. *The pseudo-BNF for a data criteria record.*

```
DataCriteriaDefn   ::= 'DATA_CRITERIA' blanks record_name
'{'
  data_criteria_defn
'}'

data_criteria_defn ::= (
   'PATH_PATTERN' blanks pattern_datas newline
|  'NAME_PATTERN' blanks pattern_datas newline
|  'LINK_PATH' blanks pattern_datas newline
|  'LINK_NAME' blanks pattern_datas newline
|  'CONTENT' blanks content_fields newline
|  'MODE' blanks mode_specs newline
```

```
|  'DATA_ATTRIBUTES_NAME' blanks name newline
   )
 pattern_datas      ::= pattern_data [('&' | '|') pattern_datas]
 pattern_data       ::= ['!'] pattern
 pattern            ::= a shell pattern matching expression, as
   defined in sh(1)
 mode_specs         ::= mode_spec [('&' | '|') mode_specs]
 mode_spec          ::= (
   type_spec [permission_spec]
|   type_spec ('&' | '|') permission_spec
   )
 type_spec          ::= ['!'] type_char {type_char}
 type_char          ::= ('d' | 'l' | 'f' | 's' | 'b' | 'c' )
 permission_spec    ::= ['!'] permission_char {permission_char}
 permission_char    ::= ('r' | 'w' | 'x')
 content_fields     ::= content_field [('&' | '|') content_fields]
 content_field      ::= (
   ['!'] offset blanks 'string' blanks string
|  ['!'] offset blanks 'byte'  blanks data_values
|  ['!'] offset blanks 'short' blanks data_values
|  ['!'] offset blanks 'long'  blanks data_values
|  ['!'] offset blanks 'filename' blanks string
   )
 offset             ::= an unsigned decimal integer
 data_values        ::= data_value [blanks data_values]
 data_value         ::= an unsigned integer: decimal, octal (leading
   0) or hexadecimal (leading 0x or 0X)
 name               ::= ( "A–Z" | "a–z" ) [name_char]
 name_char          ::= { "A–Z" | "a–z" | "0–9" | "–" }
 string             ::= a character string, not including <newline>
 newline            ::= '\n'
 blanks             ::= one or more <blank>s (spaces and/or tabs)
```

Data Attribute Records. Data attribute records define a data type's name as well as other attributes of the data type such as its icon and application binding. The data type name is the same as the data attribute record name. The following declaration shows the data attribute record for the Image data attribute defined in the data criteria records above. A data attribute record can be associated with an unlimited number of data criteria records.

```
DATA_ATTRIBUTE Image
{
  ACTIONS        Open, Print
  ICON           imagedata
  MIME_TYPE      image/jpeg
  PROPERTIES     visible
  MEDIA          Image_Data
  DESCRIPTION    Data type for the
                 ImageViewer application
}
```

This data type's icon is imagedata. Applications such as the CDE file manager will use this icon to represent image files. For example, if the file manager finds a file named kite.gif, the file will be represented with the imagedata icon. This data type also defines attributes that may be of interest to other data-typing-aware applications. For example, a mailer application may use the value of the MIME_TYPE attribute to decide how an attachment should be viewed.

Data attribute records are the only record type that can contain application-specific fields—they are not limited to a fixed set of field names. Fig. 4 shows a pseudo-BNF for data attribute records.

Fig. 4. *The pseudo-BNF for a data attribute record.*

```
DataAttributesDefn   ::= 'DATA_ATTRIBUTES' blanks record_name
'{'
                         data_attributes_defn
'}'
data_attributes_defn ::= (
```

```
                    'DESCRIPTION' blanks string newline
                  | 'ICON' blanks string newline
                  | 'INSTANCE_ICON' blanks string newline
                  | 'PROPERTIES' blanks string {',' string} newline
                  | 'ACTIONS' blanks name {',' name} newline
                  | 'NAME_TEMPLATE' blanks string newline
                  | 'IS_EXECUTABLE' blanks string newline
                  | 'MOVE_TO_ACTION' blanks string newline
                  | 'COPY_TO_ACTION' blanks string newline
                  | 'LINK_TO_ACTION' blanks string newline
                  | 'IS_TEXT' blanks string newline
                  | 'MEDIA' blanks string newline
                  | 'MIME_TYPE' blanks string newline
                  | 'X400_TYPE' blanks string newline
                  | unique_string blanks string newline
                  | '#' string newline
                    )
string            ::= a character string, not including <newline>
newline           ::= '\n'
unique_string     ::= a uniquely named string for implementation
                      extensions
blanks            ::= one or more <blank>s (spaces and/or tabs)
```

Data Typing Service APIs. Before the data typing APIs can be used, an application must first load the databases by calling the DtDbLoad function. After this, an application should register a database modification callback function using the DtDbReloadNotify function. This callback will be invoked when a user invokes the action ReloadActions to notify data-typing-aware applications that a data type or action record has been added, modified, or deleted. If an application fails to register the modification callback, it will not be notified of a database change, resulting in the application having an outdated database in memory and possibly appearing and behaving differently than applications that received the notification.

The first consideration in choosing the API to retrieve information from the data type database is whether the application already has a data type name for an object or if the object of interest is a file or a pointer to a memory buffer. For example, if an application wants to determine the icon associated with the file tanager.gif, it would first make the following call:

```
char *data_type_name = DtDtsFileToDataType
                  ("tanager.gif")
```

to determine the data type name of the file. To retrieve the icon for this data type, the application would then call:

```
char *icon_name =
   DtDtsdatatypeToAttributeValue
                (data_type_name,
                "ICON",
                NULL)
```

For the data criteria and data attribute records given above, these two sequences of calls would result in setting icon_name to imagedata.

The next consideration is whether the desired information is a specific attribute of a data type or a (NULL-terminated) list of all attribute name and value pairs for a data type. The following function retrieves all of the attributes for a memory buffer:

```
DtDtsAttribute **attributes =
            DtDtsBufferToAttributeList
               (void *buffer,
               int buffer_length,
               NULL)
```

For performance reasons, if an application needs more than one attribute for a data type, the APIs to retrieve all attributes with a single call should be used.

The Action Service

In CDE, the action service provides the infrastructure for consistent data access through polymorphic action naming. Desktop components, such as the CDE file manager and the CDE mailer, use the action service to start an application

with a user's data. When a user opens a data file in the file manager, the action service opens the application associated with the data type. For example, when a mail folder is dropped on the mail control in the CDE front panel, the action service is used to open a mail instance with the dropped folder.

The following action types are provided in the action service:

- Map actions, which provide consistent action naming
- Command actions, which encapsulate command lines
- Message actions, which use the CDE message service to encapsulate a request or notification message.

Map Actions. Map actions provide a mechanism to present action names consistently, regardless of data type. CDE provides numerous data types and each type has an associated open action (an open action usually starts the type's application). To get consistent naming, the open action for each data type is named Open and each data type's print action is named Print. This allows applications such as the file manager to include in a data file's list of actions the names Open and Print, regardless of the actual action used to open or print the data type. The following are examples of map actions.

```
ACTION Open
{
  TYPE       MAP
  ARG_TYPE   Image
  MAP_ACTION Open_Image
}

ACTION Print
{
  TYPE       MAP
  ARG_TYPE   Image
  MAP_ACTION Print_Image
}

ACTION Open
{
  TYPE       MAP
  ARG_TYPE   Gif
  MAP_ACTION Open_Gif
}
```

The TYPE attribute is MAP for map-type actions. The ARG_TYPE attribute gives the name of the associated data attributes record. The MAP_ACTION attribute gives the name of the action that will be indirectly invoked when the Open action is invoked.

The first Open and Print definitions are map actions for the Image data type defined in the Image data attribute record given above. Data attribute records and action records are linked by an action's ARG_TYPE field. If an application invokes an open action on an Image data type, the resulting action that gets invoked is Open_Image.

The second Open definition is another open action but for GIF-type files. When data of this type is opened, the Open_Gif action is indirectly invoked. Overloading map action names makes it possible to present the user with consistent action names.

Command Actions. Command actions are used to encapsulate arbitrary command lines (e.g., UNIX® commands and shell scripts) and application execution strings. Command actions, identified with a TYPE field of COMMAND, have three types of attributes: invocation, signature, and presentation. The following declaration shows these three types with a command action definition for the Open_Image action.

```
ACTION Open_Image
{
 # Invocation attributes:
 TYPE          COMMAND
 EXEC_STRING   /opt/imageviewer\
               bin/imageviewer\
               %Arg_1%
 WINDOW_TYPE   NO_STDIO
 EXEC_HOST     %DatabaseHost%,\
               mothra.x.org

 # Presentation attributes:
```

```
 ICON           imageapp
 LABEL        Image Viewe
 DESCRIPTION    Invokes the Image-\
              viewer application\
              for Image data types

 # Signature attributes:
 ARG_TYPE     Image
 ARG_COUNT      *
 ARG_CLASS      *
}
```

The invocation attributes define an action's execution parameters. The EXEC_STRING attribute contains the action's command line. The command line may contain keywords (e.g., %Arg_1%) which are replaced when an action is invoked.

The WINDOW_TYPE attribute specifies the window support required by the action. In this example, the image viewer application creates its own window so its WINDOW_TYPE is NO_STDIO. Other window types are available for commands needing terminal emulation support.

The EXEC_HOST attribute is used to specify a list of potential hosts on which the command could be executed. A keyword can be used to refer to a host name generically rather than explicitly naming a host. For example, the keyword %DatabaseHost% refers to the host on which the action is defined. When a list of hosts is used, the command will only be executed once, on the first host on which the command exists.

The presentation attributes (all are optional) are ICON, LABEL, and DESCRIPTION. Applications like the file manager use the ICON attribute to determine the icon to use to represent an action. The LABEL attribute defines a (potentially localized) user-visible name for an action. The DESCRIPTION attribute contains a short description of the action. An application should display the description as a result of a user's request for specific information about an action.

The signature attributes ARG_TYPE, ARG_COUNT and ARG_CLASS define the arguments accepted by an action. ARG_TYPE specifies a list of the data type names an action accepts. The data type names refer to data attribute record names. The ARG_ COUNT attribute specifies the number of arguments an action accepts. The ARG_CLASS attribute specifies the class of arguments the action accepts, such as files or memory buffers.

When an application invokes an action, it gives the action service an action name and data arguments. The action service first searches the action database for actions matching the action name. Next the action record that matches the action name is checked for a match between the action record's signature attributes and the data arguments. If a match is found, the associated action will be invoked, otherwise an error is returned to the application.

Message Actions. The CDE message service provides the ability for applications to send and receive messages. The action service in turn provides the ability to encapsulate a message. A message-type action is specified by setting the TYPE attribute to TT_MSG. The following is an example of a message-type action.

```
ACTION OpenDirectoryView
{
 TYPE           TT_MSG
 TT_CLASS     TT_REQUEST
 TT_SCOPE     TT_SESSION
 TT_OPERATION Edit
 TT_FILE      %Arg_1"Folder to open:"%
 TT_ARG0_MODE TT_INOUT
 TT_ARG0_VTYPE  FILE_NAME
 DESCRIPTION    Request the File\
              Manager to open a \
              user-specified folder
}
```

The TT_CLASS attribute specifies if the message is a request or a notification. A request message requires that the application that accepts the message respond to the request. A notification message may be accepted by more than one application but no response is required.

The TT_SCOPE attribute specifies the message's scope. The scope of a message is typically the user's current CDE session. However, a message can also be file-scoped. File scoping is used by applications that are interested in receiving message events for a specific file. For example, if a file-scoping-aware editor opens or saves a file, other applications that are also file-scoped to the same file will be notified of the operation.

The TT_OPERATION attribute specifies the message's operation. The operation field may be application-specific, but may also specify a generic desktop operation such as Display or Edit as defined by the message service's media exchange message set.

The TT_FILE attribute specifies a file name for a message. In the example above, this field contains a keyword that results in the user being prompted for the name of a folder to open. The user's input will be placed in the message before it is sent.

A message can have any number of arguments. The arguments can be used to give data to the servicing application or to specify that data should be returned to the requesting application. The attributes TT_ARGi_MODE and TT_ARGi_VTYPE specify the input/output mode and type of a message for the ith message argument.

Actions, Data Typing, and Drag and Drop. The data typing and action services can be combined to define the drag and drop semantics for a data type. The following data attribute definition defines its default action (the action to be invoked when data of this type is opened) as OpenAction. This definition uses the MOVE_TO_ACTION, COPY_TO_ACTION, and LINK_TO_ ACTION attributes to define actions to be invoked when data of this type is moved, copied, and linked respectively.

```
DATA_ATTRIBUTE DragAndDropAwareType
{
  ACTIONS           OpenAction

  MOVE_TO_ACTION    MoveAction
  # Move = Shift + Mouse Button 1

  COPY_TO_ACTION    CopyAction
  # Copy = Control + Mouse Button 1

  LINK_TO_ACTION    LinkAction
  # Link = Control + Shift + Mouse Button 1
}
```

Action Service APIs. The function used to invoke an application, DtActionInvoke, has several arguments. However, the parameters described here include the name of the action to invoke, an array of data arguments for the action, the number of arguments, and a callback function to be invoked when the action terminates.

The action name is used to find action records with the same name in the action database, and the signature attributes of the matching action records are searched to find a match with the API's data arguments. If a matching action is found, the action is invoked and an action invocation identification number is returned to the application, otherwise an error is returned to the application. The data arguments can be files or memory buffers. If an action is defined to accept zero or one arguments but more than one argument is provided, an instance of the action will be invoked for each argument.

To be notified when an action terminates, an application can register an action termination callback when the action is invoked. This is particularly useful for applications that invoke actions on behalf of embedded objects. For example, the mailer uses this feature to get notified when its attachments, which have been opened by an action invocation, are closed. If an action has more than one action instance outstanding, it can use the action invocation identification number to determine which action instance terminated.

When a data argument is a memory buffer, the application must provide a pointer to the buffer, the size of the buffer, and an indication of whether the action is allowed to modify the buffer. Additionally, the application can provide the data type of the buffer and a file name template to be used if a temporary file for the buffer is needed. When necessary, such as when a buffer is writable, the buffer is copied to a temporary file and the file name is used in the action invocation. When the action terminates, the contents of the file are copied to the buffer, and the temporary file is removed.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open is a registered trademark and the X device is a trademark of X/Open Company Limited in the UK and other countries.

▶ Go to Article 4

▶ Go to Table of Contents

▶ Go to HP Journal Home Page