Creating DIY Multi-Tasking

David Spencer shows how Acorn s Desktop Development Environment can be extended to other uses, and all necessary modules are on the magazine

Acorn's DDE, as reviewed in RISC User Volume 4 Issue 9, is a set of linked language development tools all running within the Desktop environment. This set includes the C compiler, ARM assemblers, the linker and so on. These tools all have one factor in common - they were originally designed to run as single-tasking command line applications. In order to make them multi-tasking, Acorn developed a method that would allow them to be started from the Desktop, setting up options through a dialogue box, and run in a window, with any output produced saved with a normal save box.

Although this multi-tasking 'front-end' was designed for Acorn's compilers and language tools, it can equally be applied to any command line driven application or utility, both commercial or home produced. In this article we shall show exactly how this is done.

There are two modules supplied with the DDE that are responsible for the multi-tasking tools. The first of these is called *TaskWindow*, and this allows an otherwise non-multitasking application to be pre-emptively multi-tasked, with all its output being sent to a window. It is a version of this module that is used by *Edit* to create 'Task Windows'. The second module, and the one of most interest to us here, is called *FrontEnd*. The purpose of this module is to provide an icon on the icon bar for the tool, and allow the command line parameters to be set up by way of a dialogue box, and optionally via a menu as well. *FrontEnd* will then handle the calling of the *TaskWindow* module, and provide a window to display the output. Finally, it generates the save box needed to save any output file produced by the tool.

WHAT CAN IT DO?

Before continuing, it's worth noting exactly what type of applications, or tools, can be run in this way. Essentially, the tools need to be non-interactive, which means that once started by a star command, they do their job without requiring any further action from the user, produce their output, and then exit back to the command line. Additionally, if they produce file output, then only one object should be produced. This means that most language compilers and other file 'processors' are eligible, but the likes of text editors obviously aren't.

Some applications offer a mixture of interactive and non-interactive operation. For example, the Acorn ARM Assembler and ChangeFSI will both run non-interactively, but enter an interactive mode if all the necessary information is not given on the command line. This class of tools is OK, because you can easily ensure that all the necessary information is given up front.

FrontEnd allows a dialogue box to be opened, either by clicking on the tools icon bar icon, or by dragging a file to it. Options can then be set in this dialogue box, and through its associated menu, and the tool run. This setup process can optionally be bypassed and the tool automatically run when a file is dragged to it, and similarly, the output file can be saved automatically rather than producing a save box. The layout of the dialogue box is determined by the template that you provide, while the relationship between this and the command line parameters, as well as the menu entries, are controlled by a descriptive text file included for each tool.

Each icon in the dialogue box can add a parameter to the tool's command line. Taking the C compiler as an example, any text within the 'Include' writable icon is added to the command line preceded by the '-I' qualifier that tells the compiler this is a list of *include* paths. This is an example of an icon adding a combination of a fixed part ('-I') and a variable part (the path list) to the command line. On the other hand, if the 'Compile only' icon is enabled this just adds the fixed string '-c' to the command line. In other words, this is a simple on-off qualifier. Menu entries also add to the command line if they are enabled. For single entries they add a fixed string, while entries with writable sub-menus can add the contents of that entry, optionally preceded by a fixed string.

As well as adding to the command line, icons can be made to increase or decrease the numeric values of other icons. This is normally used for up and down arrow icons that modify a value, as in Acorn's *Common* tool. Enabling an icon can also change the dialogue box itself. An example of this is the 'Wildcards' icon in the *Find* tool. When enabled, this extends the dialogue box to display a set of icons representing the available wildcards. As a further twist, you can specify that selecting a particular icon will deselect another if it is currently selected. This is used to prevent clashes between mutually exclusive options, such as 'Compile only' and 'Preprocess only' in the C Compiler. In a similar vein, selecting an icon can grey-out another one. An example of this is in *LibFile*, where selecting the 'Create' icon greys out the writable box for the library name, as a name isn't applicable in this case.

The description file for each tool must define precisely how the dialogue box and menu work, and to achieve this, its structure is rigidly defined in the DDE User Guide by means of a description language called Extended Backus-Naur Form (EBNF). This defines the exact syntax and semantics for the tool's description, but doesn't define the pragmatics (i.e what each entry actually does). EBNF is easily understood, but can be daunting if you have never met it before. The best way to study it is to look at the description file for a particular tool (the file 'Desc' within the tool's application directory) and compare that against the EBNF definition.

USING FRONTEND

To use *FrontEnd* with a new tool, all you need to do is to build an application directory for the tool and provide the appropriate description file. To show how this is done, the remainder of the article will concentrate on producing a very simple tool that will merely display its command line, so that you can see what is going on.

A PRACTICAL EXAMPLE

To start with, create an application directory called *!Example*, and copy into it the files *Messages* and *Templates* from the C Compiler's directory, !CC. Next, create a *!Sprites* file containing a suitable sprite called !example. If you want, you can use the standard screwdriver and scanner tool icon by copying the !Sprites file from !CC, but remember to rename the sprite itself.

The *Templates* file contains a total of nine templates, eight of which are standard for all *FrontEnd* tools and shouldn't be changed. The ninth, 'Setup', is the main dialogue box template. For the purposes of this example, delete the existing 'Setup' template and create a new one. Note that for all tools, icons zero and one must be menu icons containing the text 'Run' and 'Cancel' respectively, and the entry for icon zero must be indirected. In the new template, make icon two writable, icon three an option button and icons four and five radio icons. You can call the icons whatever you want, but the option and radio buttons must have the button type 'Click'.

All applications require a *!Run* file, and again this is best copied from !CC, although you will need to change the line that sets 'CC$Dir' to set 'Example$Dir' instead, and change the final line to read:

```
    *FrontEnd_Start -app Example -desc
<Example$Dir>.desc
```

The run file includes the RMEnsure commands needed to load the modules required for *FrontEnd* to operate, and then starts *FrontEnd* using the command *FrontEnd_Start. This takes two parameters, the first is the name of the tool, and the second the filename of the description file that we are about to write.

Before writing the description file, we need to design the command line format, though for existing applications of course this is already determined. Quite arbitrarily, our command line format will be, together with the icons that control it, an input filename (icon 2), and output filename preceded by '-out', and three qualifiers, '-faster' (icon 3), '-onepass' (icon 4) and '-twopass' (icon 5). The last two of these are mutually exclusive. We'll also allow two further parameters to be set from a menu, '-noerrors' and '-listfile', the second of these being followed by a filename. For example, a typical command line may look like this:

```
    *example infile -out outfile -faster -onepass -
listfile$.list
```

Note that we do not need to specify the output filename in the dialogue box, as the file will be saved using a standard save box which allows the name to be set anyway.

A suitable description file is given in listing 1. This should be entered as a text file using Edit and saved with the name *Desc* within the *!Example* directory. This file can be split into a number of distinct sections, each with appropriate keywords marking the start and end.

The first section, the tool details, defines the name of the tool, the command used to run it, the version number and the wimpslot needed. The second section specifies that a window is required to display the output from the program as it is running. The third section specifies that the output filename must be preceded on the command line by '-out'. What happens is that *FrontEnd* provides a temporary filename on the command line, and when you save the output using the save box it copies this to the final destination. The next section controls the dialogue box, and is split into three sub-sections. The first of these specifies that the string in icon two is added directly to the command line, whilst icons three, four and five add the specified qualifiers to the command. This is followed by a sub-section defining the defaults for the icons, and a further one controlling what happens when files are dragged in. In this case, files dragged to the icon bar are put into the writable icon replacing anything already there, whilst files dragged to icon two (the writable icon) are added to it using space as a separator.

The fifth section performs a similar task, but for the menu. It lists each entry name together with the string it maps to, and the title for the sub-menu and its length in the case of the second entry. Again, a second sub-section specifies the default states. The final section controls which icons deselect which others. In this case, it is set so that the '-onepass' and '-twopass' qualifiers cannot be active together.

All that remains now is the example program itself. This is very simply:

```
10 REM >Example
20 SYS "OS_GetEnv" TO A$
30 PRINT A$
40 END
```

This file should be saved as *Example* within the application directory. You are now in a position to try out the new tool, and see what effect changing the icons and menu selections has on the command line.

WHAT NOW?

The combination of this introduction and overview together with the DDE User Guide and the examples in the form of the tools supplied with the DDE, should allow you to incorporate *FrontEnd* into any suitable application. This is not only useful to add tools to the DDE, but could also be used totally separately. Indeed, you could licence the *FrontEnd* module from Acorn and use it as part of a commercial application.

The modules TaskWindow, FrontEnd and also DDEUtils are included under licence from Acorn on this month's magazine disc.

Listing 1. The description file

```
tool_details_start
  name          "Example";
  command_is    "<Example$Dir>.Example";
  version       "1.00";
  wimpslot      32k;
tool_details_end

metaoptio
ns_start
  has_auto_save    ^.leafname from icn 2;
  has_text_window;
  has_summary_window;
metaoptions_end

fileoutput_start
  output_option_is  "-out";
  output_dft_is     produces_output;
fileoutput_end

dbox_start
  icons_start
    icn 2  maps_to string;
    icn 3  maps_to "-faster";
    icn 4  maps_to "-onepass";
    icn 5  maps_to "-twopass";
  icons_end

  defaults
    icn 3  off,
    icn 4  off,
    icn 5  on;

  imports_start
    drag_to icn 2 inserts icn 2 separator_is " ";
    drag_to iconbar inserts icn 2;
  imports_end
dbox_end

menu_start
  "No errors"      maps_to "-noerrors";
  "Listing"        maps_to "-listfile " sub_menu "   List file:   "
256;

  defaults
    menu  1 off,
    menu  2 off;
menu_end
```

```
deselections_start
    icon 4 deselects icon 5;
    icon 5 deselects icon 4;
deselections_end
```