AmigaMail

COLLABORATORS

	TITLE : AmigaMail	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 23, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 AmigaMail **1**

1.1 II-1: Executing External Programs with Release 2.0 1

1.2 The System() Function 1

1.3 The 2.0 Con-handler 3

Chapter 1

AmigaMail

1.1 II-1: Executing External Programs with Release 2.0

by Carolyn Scheppner

The 2.0 Amiga operating system provides more flexible methods of launching programs from an application. The 2.0 dos.library contains a new function, System(), which is more flexible and powerful than the 1.3 Execute() function. Improvements to the con-handler, CON:, allow programs to create ``auto con'' windows, which are console windows that will only appear if input is requested from or output is sent to that console. The 2.0 con-handler also provides several other enhancements, including Close gadgets for CON: windows and a feature which allows a program to use an existing window as its CON: window.

The System() Function
The 2.0 Con-handler

1.2 The System() Function

The System() function is intended to replace the Execute() function of release 1.3. System() spawns a shell process to execute the command line which is passed to System() as an argument. The shell parses the command-line normally, just like command-lines typed directly into the shell's console. If it can, System() will pass the current path and directory to the programs it launches.

The System() function can execute external commands either synchronously or asynchronously. When System() is used synchronously, control returns to the calling program after the external program has completed. In this case, System() returns the external program's return code, or a -1 if the command could not be found or run. On the other hand, when System() initiates a program asynchronously, the program is no longer a concern for the caller. The operating system will take care of the cleanup. This is extremely useful for an application that must start up multiple programs on user demand, such as a hot key commodity. By default, System() starts programs

synchronously. To launch a program asynchronously, use the SYS_Asynch tag with the data field set to TRUE.

With the System() call, it is easy to provide programs with specific input and output handles. The tags SYS_Input and SYS_Output (defined in dos/dostags.h) are used to supply the input and output file handles. A program can pass its own input and output handles to a synchronously launched program by passing the results of Input() and Output(), respectively. Note that with a synchronous System() call, the OS will not close these handles when the spawned process exits. In the case of an asynchronously launched program, the launching program normally must provide new IO handles since the system automatically closes these handles when the asynchronous process ends. Because AmigaDOS wants separate handles for input and output, System() will automatically create an output handle if it's passed a handle for SYS_Input and NULL for SYS_Output. This allows a program to open a CON: window for input and use it for both input and output, as System() will test to see if input is interactive and, if so, will attempt to open ''*'' for output to that console. If the input file handle is not interactive, System() opens ''*'' on the current console task.

Programs launched using System() are not restricted to the built-in, or Boot, shell. In the near future, System() will be able to take advantage of specially designed shells (the design requirements of these special shells have not yet been documented). Two tags, SYS_UserShell and SYS_CustomShell, specify which shell System() should use to execute the command-line. By using the SYS_UserShell tag with a tag value of TRUE, an application tells System() to send a command to the user's preferred shell rather that the boot shell (Note that the default user shell is the boot shell). If an application opens a shell for the user or executes the user's command-lines, it should tell System() to use the user shell. If an application requires consistent shell behavior, it must not use the user shell because the user can change the user shell. Another shell tag, SYS_CustomShell, allows an application to choose other shells besides the boot and user shells. This tag's data field should contain the custom shell's name as it appears in the system resident list.

Although it does offer many features, System() does have some limitations. First, because command paths currently only exist in the CLI structure, Workbench (non-CLI) processes have no paths. Consequently, when a Workbench process calls a program using System(), the shell has no path to search for that program (aside from the system default search path of C: and the current directory). A second limitation of the System() function involves CTRL-C/D/E/F handling. When a task opens a CON: window to provide a handle for a System()-launched command, that task is the owner of the handle. This has two effects. A System()-launched command running in that CON: window will only be able to receive CTRL-C/D/E/F signals while it is doing input or output to the window (i.e. when it has a pending read or write). If the task that owns the handle is still around, it receives CTRL-C/D/E/F signals whenever those keys are pressed in the CON: window.

1.3 The 2.0 Con-handler

The 2.0 con-handler (CON:) has many new enhancements that allow programs to further customize their console windows. An application requests these new features by appending keywords to the end of the CON: specification string for Open(). These keywords may appear in any order after the title string in the CON: specification.

These new keywords are:

AUTO		Don't open CON: window until/unless input or output occurs
CLOSE		Put a close gadget on the CON: window
WAIT		Hold off Close until user clicks Close or types CTRL-\
WINDOW	0xaddr	Use specified window (may be on a custom screen)
SCREEN	name	Open on specified public screen

The additional CON: keywords BACKDROP, NODRAG, NOBORDER, NOSIZE, SIMPLE, and SMART, allow control of other attributes of a CON: window.

An AUTO/CLOSE/WAIT CON: window is perfect for 2.0-specific C startup code and for asynchronous System() startup of arbitrary commands. Because of the ``auto con'' feature (AUTO), the system will never open the CON: window of a program that doesn't do any stdio (example: Calculator). If a command does stdio input or output, the window will not open until stdio occurs. If the window opens, the wait feature (WAIT) causes the window to stay open until the user types CTRL-\ (end of file) or clicks the Close gadget. The CLOSE keyword tells the con-handler to put a close gadget on the CON: window.

Example: ``CON:/0/0/640/200/My Title/AUTO/CLOSE/WAIT''

The SCREEN keyword along with a public screen name allows an application to open a CON: window on that public screen. Alternately, an application can use the WINDOW keyword to attach a console to an already open Intuition window. The hex address of the Intuition window must follow the WINDOW keyword. This makes it possible for System()-launched programs to do their stdio in a window on a custom screen.

The example code, SystemTest.c, provides two simple subroutines for executing external commands, and demonstrates the following 2.0 features:

- Synchronous System() command execution using the calling program's Input()/Output().
- Synchronous System() command execution in a custom screen window.
- Asynchronous System() command startup with an AUTO/CLOSE/WAIT window.
- OpenScreenTags and OpenWindowTags for a 2.0 New Look window.