#13: OOPTESample

Written by: Keith Rollin

Versions:	1.00	April 1989
Components:	`BuildOOPTESample`	April 1, 1989
	`MOOPTESample.p`	April 1, 1989
	`OOPTESample.make`	April 1, 1989
	`TECommon.h`	April 1, 1989
	`TESampleGlue.a`	April 1, 1989
	`TESample.r`	April 1, 1989
	`UApplication.p`	April 1, 1989
	`UApplication.inc1.p`	April 1, 1989
	`UDocument.p`	April 1, 1989
	`UDocument.inc1.p`	April 1, 1989
	`UTEDocument.p`	April 1, 1989
	`UTEDocument.inc1.p`	April 1, 1989
	`UTESample.p`	April 1, 1989
	`UTESample.inc1.p`	April 1, 1989

The build process for OOPTESample is entirely automated. All you need to do is run the BuildOOPTESample script. BuildOOPTESample is a variation on the BuildProgram script that comes standard with MPW. It creates a folder to contain the intermediary object files, and then calls Make with the file OOPTESample.make. Make's output is executed with the final application OOPTESample as the result.

OOPTESample is an example application that demonstrates how to initialize the commonly used Toolbox managers, operate successfully under MultiFinder, handle desk accessories, and create, grow, and zoom windows. It demonstrates fundamental TextEdit toolbox calls and TextEdit automatic scrolling, and it shows how to create and maintain scroll bar controls.

This version of TESample has been substantially reworked in Object Pascal to show how a "typical" object-oriented program could be written. To this end, what was once a single source code file has been restructured into a set of classes which demonstrate the advantages of object-oriented programming.

There are four main classes in this program. Each one of these has an interface (.p) file and an implementation (.inc1.p) file, and is compiled into its own separate `UNIT`.

The TApplication class does all of the basic event handling and initialization necessary for Macintosh Toolbox applications. It maintains a list of TDocument objects and passes events to the correct TDocument class when appropriate.

The TDocument class does all of the basic document handling work. TDocuments are objects that are associated with a window. Methods are provided to deal with update, activate, mouse-click, key-down, and other events. Some additional classes which implement a linked list of TDocument objects are provided.

The TApplication and TDocument classes together define a basic framework for Macintosh applications, without having any specific knowledge about the type of data being displayed by the application's documents. They are a (very) crude implementation of the MacApp application model, without the sophisticated view hierarchies or any real error handling.

The TESample class is a subclass of TApplication. It overrides several TApplication methods, including those for handling menu commands and cursor adjustment, and it does some necessary initialization. Note that we only need to override nine methods to create a useful application class.

The TEDocument class is a subclass of TDocument. This class contains most of the special-purpose code for text editing. In addition to overriding most of the TDocument methods, it defines a number of additional methods which are used by the TESample class to get information on the document state.

This program consists of four segments. "Main" contains most of the code, including the MPW libraries and the main program. "Initialize" contains code that is used only once, or rarely, and can be unloaded after the event loop is completed. "%A5Init" is automatically created by the Linker to initialize globals for the MPW libraries and is unloaded right away. "%_MethTables" is a fake segment used by Object Pascal to maintain object relationships.

Toolbox routines do not change the current port. In spite of this, in this program we use a strategy of calling `_SetPort` whenever we want to draw or make calls which depend on the current port. This precaution makes us less vulnerable to bugs in other software which might alter the current port (such as the bug (feature?) in many desk accessories which changes the port when there is a call to `_OpenDeskAcc`). Hopefully, this also makes the routines from this program more self-contained, since they don't depend on the current port setting.

This program does not maintain a private scrap. Whenever a cut, copy, or paste occurs, we import or export from the public scrap to TextEdit's scrap right away, using the `TEToScrap` and `TEFromScrap` routines. If we did use a private scrap, the import or export would be in the activate or deactivate event and suspend or resume event routines.