Memorandum

Sage Source-Code Instructions

To: Sage Application Programmers
From: David Gedeon
April 13, 2006

License

The purpose of the source-code distribution is for you to customize the Sage executable code for use at your organization. Code development is not restricted to a single computer but, normally, the resulting executable code remains for single-computer use. Multiple-computer installation of your executable code requires a site license.

Under no circumstances are you allowed to sell or distribute your custom Sage executable application outside your development site. While you do own the copyright for model-component software created entirely by you, and conceivably could freely distribute this component-specific software, Gedeon Associates still owns the copyright for any original Sage software linked in with your entire application.

Installation

The source code files and directory structure are self-installing using the Setup.exe program on the distribution CD. By default they are installed under the directory c:\Program Files\Gedeon\Sage[n]\Source, where [n] is the version number. You should copy the entire Source directory to another location for development use. The subdirectory structure under Source is important because the search paths in the Delphi project files are set up for the existing structure.

Files and Directories

The distribution files for the different model classes consist of a Delphi project file along with a "splash" unit that performs certain initializations and contains the bitmap that appears when the Sage application loads and in the Help|About dialog. These are located under the SageApps directory.

SageApps subdirectory	Delphi Project file	Splash/Initialization unit
Stirling\Appl	Stirling.dpr	StlSplsh.pas
Ptube\Appl	Ptube.dpr	PtbSplsh.pas
LowTCooler\Appl	LTCooler.dpr	LTCSplsh.pas

The root-model unit defines the highest-level model in each model class.

Subdirectory	Root model unit
Stirling\Models	StlRoot.pas
Ptube\Models	PtbRoot.pas
LowTCooler\Models	LTCRoot.pas

Other supporting files of a more generic nature are in the following subdirectories:

\bin	destination for exeutable files
\dialogs	dialog box related
\models	model-specific source files; identifier include files; bitmaps; Windows resource files
\units	similar to \models, but for general purpose source files
\propbase	gas and solid property database manager files

The above subdirectories appear under the SageApps directory, under each model-class (Stirling, Ptube, LowTCooler). Some of them also appear directly under the Source directory for files common to all Sage applications.

Compiling

To compile and run the project you will need a copy of the Borland Delphi compiler.

To create a custom application, open one of the above project files under the Delphi IDE and use the "SaveProjectAs" command to save it under a different name, to distinguish it from the offical Sage application. Likewise "SaveAs" the associated splash unit (e.g. StlSplsh.pas) and root-model (e.g. StlRoot.pas) under new names.

Only the visual forms used by Sage applications are included in Sage project files (.dpr) and managed by Delphi's Project Manager. Nonvisual units and other Sage ingredients are included through "uses" statements in unit interfaces and $I include directives. The only exception to this is the application root-model unit (e.g. StlRoot.pas) which is included to simplify creating custom applications when you "SaveAs" under a different name.

Visual forms are automatically created by Delphi project-management mechanisms. If they were not included in the .dpr file then they would have to be explicitly created and destroyed within the application. Nonvisual units just clutter up Project Manager's window and result in no actual code generation. So they are omitted.

Project Options

As installed the settings of the "Project|Options" dialog should be correct. Directory search paths are required to find progam "units" referenced by the 'uses" statements in other units, outside the scope of Delphi's Program Manager. If the compiler has problems finding these units you will have to update the Directories search path (under "Directories/Conditionals" tab). While you are

at it, under the "Compiler" tab, make sure the the "Extended syntax" and "Assignable typed constants" boxes are checked. Under the "Compiler Messages" tab, uncheck the "Unsafe type", "Unsafe code", and "Unsafe typecast" boxes to avoid a lot of messages for code now considered unsafe as of Delphi 7, for applications (unlike Sage) intended for Microsoft's .NET platform.

Digging In

There will no doubt be some initial shock and heart palpitation in confronting for the first time the large number of files in the source-code distribution. But there is logic and order here and most of the files can be initially ignored. The following table covers the files at the highest level of organization:

file extension	description	notes
*.pas	Pascal unit	Generally heavily documented internally. Files related to model component physics contain "mdl" in the name, e.g. gasmdl.pas. Files related to boundary connections contain "cnct" in the name, e.g. flowcnct.pas
*.dcu	compiled unit	Compiler generated from *.pas file
*.dfm	binary form	Compiler generated from *.pas file
*.inc	include file	Windows resource identifier constants.
*.sid	include file	Stream identifier constants. Identify model components and variables read from an input file.
*.rc	resource	Windows resources such as icon bitmaps
*.res	resource	Compiled *.rc file for linking into executable file; requires a program like Borland's Resources Workshop to generate
*.bmp	bitmap	Mainly model component icons.

Files named *mdl.pas files, contain model component physics. Simplest to understand are the gizmo related model components comprising the basic moving parts, springs and dampers of the Sage component library. The ancestor moving part component, from which all others derive, resides in movmdl.pas and its various derivatives reside in sprmdl.pas, flxmdl.pas, dmpmdl.pas, rcpmdl.pas and pismdl.pas. At the other end of the complexity spectrum are the gas-domain model components in gasmdl.pas with descendant classes in cgasmdl.pas, dgasmdl.pas, mgasmdl.pas.

All model-components share common elements. If you are interested in how things get connected together you will want to look at cnctobj.pas, and its derivatives forccnct.pas and prescnct.pas, which deal with force and pressure connections specifically. For an understanding of how grids work, you will want to look at cmpgrids.pas. For delving into the ancestral class types for variables and model components, look into model.pas. You will find additional variable classes in VarObj.pas, and the classes which orchestrate solving, mapping and optimiza-

tion in Process.pas. The classes that define the model-component palette and the process of giving birth by drag-and-drop are in Palette.pas.

Customizing Model Components

Depending on what you want to do, this can be either easy or quite involved. At the easy extreme, you might want to slightly revise the Pascal coding for an existing model component. In this case you would first find the desired function in a *mdl.pas file, make the changes and re-compile. At the other extreme, you might want to create an entirely new model component. The best way to proceed here would be to first copy the code for the closest available model component (maybe an entire unit) then make required changes with heavy use of the search/replace function of the Delphi editor. My practice is to make all changes in a separate file until I'm reasonably sure they are correct, then copy the various fragments to their required locations.

A potentially confusing aspect of coding new model components is dealing with global resource and stream identifier constants which you will find in many places. These variables begin with:

s_	resource string constants
idb_	bitmap identifiers
sid_	stream type identifiers

Model-component stream identifiers reside in file MdlObj.sid. Streamable model components are registered with these identifiers in the initialization section of the unit in which they are defined. New model components require unique identifiers to avoid type conflicts during input file operations. There are also stream identifiers for connection classes CnctObj.sid and Sage classes Sage.sid, Streams.sid, but normally you can ignore these.

Model-component bitmap identifiers reside in file MdlObj.inc with associated Windows resources (bitmaps), in MdlObj.rc. These get compiled into a binary MdlObj.res file using a program like Borland's Resources Workshop. There are also resources for connection classes (CnctObj.inc, rc) and Sage classes (Sage.inc, rc) as well as those of the root-component (StlRoot.inc, rc), but you will probably not have to mess with these. The thing to remember about Windows resources is to compile them into *.res files before compiling and linking your application.

Once you have successfully created a new model component, you still have to install it in the palette of the root-model component (in StlRoot.pas) or some other component before you will be able to make use of it. This is easy, requiring only that you add a InsertSeed call to the FillChildOvary method of the parent component, following the examples already there.

In the event you are coding a new model component that differs in some fundamental way from an existing model component, you will have problems of a qualitatively different nature — namely, figuring out how the code actually works. This may take a while, as you patiently trace through function calls, reading in-line comments as you go. Perhaps this would be a good time to request assistance from Sage headquarters.