Sage Source-Code Instructions

David Gedeon
February 12, 2025

Installation

The source code files are installed in the following sub-directories under the directory C:\Program Files (x86)\Gedeon\Sage[n], where [n] is the version number.

\Docs
- \SourceCodeInstructions
  - This document

\Source
- \ Units
  - Classes providing Sage functionality for model-components, variables, memory management, calculations, solving, optimizing, file loading and storing, etc. Including the Sage GUI interface itself.
- \ Models
  - Computational grids and connector classes of general use to all model classes.
- \ Dialogs
  - Dialog forms for entering variable values, selecting model components and variables for mapping or optimization, etc.
- \ Templates
  - Sample units for creating or modifying model components, adding new variables, etc.
- \ SCFusion
  - \ Models
    - Model-specific classes providing the functionality for individual model components.
  - \ Dialogs
    - Dialog forms for the SCFusion model-class Options dialog and tabular gas properties editing and plotting.
  - \ Appl
    - Delphi project file, root-class unit defining the top-level SCFusion model class, splash unit that initializes the program state, etc.
  - \ PropBase
    - Source files for the SCFprop material property management utility.
  - \ RefpropToSage
    - Source files for the RefpropToSage utility for creating tabular gas class instances using the Refprop DLL.

Compiling

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

To create a custom application, open the SCFusion.dproj project file 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 (SCFSplsh.pas) and root-model (e.g. SCFRoot.pas) under new names.

Only the visual forms used by Sage applications are included in Sage project file 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.

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 compiler and other project settings (contained in the SCFusion.dproj file) should be correct. Except you will have to update the compiler search paths ("Project|Options" dialog), required to find program "units" referenced by the 'uses" statements in other units. 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 for applications (unlike Sage) intended for Microsoft's .NET platform.

Digging In

There may be some initial shock 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 ignored initially. 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.
*.rc	resource	Windows resources such as icon bitmaps
*.res	resource	Compiled *.rc file for linking into executable file
*.bmp	bitmap	Mainly model component icons.

Files named *mdl.pas files, contain model component physics. Simplest to understand are the 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 optimization 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

As of Sage v 13 (2025) Model-component stream identifiers are the actual class names for the model component. Streamable model components are registered with these identifiers in the initialization section of the unit in which they are defined. 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. 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. Delphi automatically compiles resources into Windows *.res files.

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. Files in the \Templates directory provide some guidelines that may be useful.