Superbase NG Quick Start Guide

A Fast Introduction to Development with Superbase NG

Neil Robinson

Superbase NG Quick Start Guide: A Fast Introduction to Development with Superbase NG

by Neil Robinson

Table of Contents

Important ix
  Copyright Information ix
  Disclaimer ix
  New Versions of this Document ix
  Software Used ix
1. Introduction 1
  Who Should Read This Book 1
  Superbase NG and SIMPOL 1
  Conventions Used in This Book 1
  Why SIMPOL? 1
  Running Superbase NG Programs 2
  Deploying Superbase NG Programs 2
  Summary 3
2. What's in the Package? 5
  Overview of the Product 5
  The Superbase NG IDE 5
  Superbase NG Personal 5
  C-Language Components and Runtime Files 5
  SIMPOL Language Libraries and Samples 6
    SIMPOL Language Libraries (`*.sml`) 7
    Supplied Superbase NG Projects 11
    Documentation 18
    Utilities 19
    Superbase Conversion Utilities 19
3. Getting Started 21
  The Essentials 21
  Preparing Our Environment 21
  Command Line Programs 22
  Dialog-Style Programs 22
  Database GUI Applications 22
  Web Server Applications 22
  Server Applications 23
  Converting from Superbase 23
4. Command Line Programs 25
  Building a Command Line Program 25
  First Steps 25
  Understanding the Code 28
  Running Our Project 28
  Improving Our Program 28
  Running the Final Version 31
  Summary 32
5. Dialog-Style Programs 33
  What's a Dialog Program? 33
  The Sample Program 33
  Creating the Project 33
  Creating the Design 34
    Setting the Stage 36
    Adding the Controls to the Form 36
    Cleaning Things Up 40
    Saving the Form 44
  Adding the Form Source to the Project 44
  Setting Up the Program 46
  Getting the Basic Form Running 47
  Finishing the Color Lab Program 49
  Summary 52

6. GUI-Style Database Programs 53
- Introduction 53
- Creating the Project 53
- Create the Database 53
- Building the Form 57
- The Program Code 60
  - The `main()` Function 61
  - The addressbookapplication Type 62
  - The Remaining Initialization Code 65
  - Preparing the Form 69
- The Finished Product 70
  - A Word About Linux 71
- Summary 72
- Advanced Topics 73

7. SIMPOL Business 75
- Introduction 75
- Special Features 75
- Working With the dataform1detailblock 76
  - About the Design of Detail Blocks 76
  - Adding New Records to Detail Blocks 76
  - Editing Records in a Detail Block 78
  - Deleting Records in a Detail Block 79
- Using the `drilldown()` Function 80
- Storing Data Correctly in Modern Windows Systems 82
- Summary 83

8. SIMPOL Server 85
- About the SIMPOL PPCS Server Programs 85
- The Configuration File 85
  - The `[Server]` Section of the Config File 85
  - The `[Files]` Section of the Config File 86
- Working with `simpolserver.exe` 87
- Working with `guisimpolserver.exe` 88
- Running `simpolserver.exe` as a Service 88
- SIMPOL Server Summary 90

9. Web Server Programs 91
- Introducing World Wide Web Server Programming 91
- Styles of Web Server Application 91
- Valuable Reading References Within the SIMPOL Documentation 92
- Sample Web Server Applications Shipped with SIMPOL 92
- The Web Server "Sandwich" Method 93
- Web Server Application Summary 93

10. Server Programs 95
- About Server Programs 95
- Accessing Server Programs 95
- Sample TCP/IP Server and Client Programs 95
- Server Programs Conclusion 95

11. Converting Legacy Superbase 97
- Where to Begin? 97
- How Superbase NG Differs 97
- So What's the *Good* News? 99
- Converting Superbase Databases to Superbase NG 99
- Converting the Forms 101
- Creating the Application 103
- Summary 104

List of Tables

1.1. Superbase NG Runtime Loaders 2
1.2. Superbase NG Runtime Components 2
5.1. Control Naming Conventions 38
5.2. Event Handling Function Naming Conventions 48

List of Examples

4.1. Initial version of `urldump.sma` 27
4.2. Updated `usage()` Function 29
4.3. The parameters Type 29
4.4. The Final Version of the `main()` Function 30
4.5. Sample Command 31
5.1. The `main()` Function of the colorlab Program 46
5.2. The remaining Empty Functions of the colorlab Program 47
5.3. The Full Implementation of the `adjustformcolorvals()` Function 49
5.4. The Code for the `hexval_olf()` and `decval_olf()` Functions 50
5.5. Handling the Events for the Color Edit Controls 50
5.6. Handling the Scroll Bar Events 51
5.7. Extracting the Digits from String Values 52
6.1. The `main()` function of the program 61
6.2. The addressbookapplication type 62
6.3. The application type 62
6.4. The Code to Create a New addressbookapplication 63
6.5. The Code for the Menu Bar 65
6.6. The Code for the Menu Bar 66
6.7. The Code for the Tool Bar 66
6.8. The Code for the Tool Bar Combo Boxes 68
6.9. The Code for the Tool Bar Initialization 69
6.10. The `prepaddressbookform()` Function 69
6.11. The `ab_onnewrecord()` Function 70
7.1. The `addorderline()` function of the SIMPOL Business program 77
7.2. The `addorderline()` function of the SIMPOL Business program 78
7.3. The `addorderline()` function of the SIMPOL Business program 79
7.4. The `findcustomer()` Function for the Orders Form 81
7.5. The `findcustomer()` Function for the Orders Form 82
8.1. A Sample SIMPOL Server Configuration File 85
8.2. A Sample `svcrunnr.exe` Configuration File 89

Important

Copyright Information

Disclaimer

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

New Versions of this Document

Newer versions of this document will undoubtedly be released from time to time. It is recommended that you always ensure that you have the latest version of the documentation. Normally the latest version will be included with any update of the main product.

Software Used

This book was written using DocBook v5. It was initially written and edited in the <oXygen/> editor. A single source in XML is used to produce the book in HTML, HTML Help, and in PDF formats.

Chapter 1. Introduction

This book provides a quickstart guide for doing development using Superbase NG. It is intended as a quick introduction to the various components that make up the Superbase NG package, as well as providing a number of introductions for various methods of using the product.

Who Should Read This Book

If you are just getting into Superbase NG development and want a high-level introduction to the parts that come with the Superbase NG product, then this book is a good place to start. It begins by describing the various pieces of the product and then goes on to explain various approaches to working with the package depending on the type of project the user is planning.

Superbase NG and SIMPOL

The name of the product is Superbase NG. That includes the development environment, Superbase NG Personal, the compiler, the code libraries, runtime system, etc. The name of the programming language is SIMPOL. As such, you may see things talk about SIMPOL instead of Superbase NG but they will be normally discussing the language, not the product. Still it is safe to think of the two as being almost interchangeable. Much of the Superbase NG product is built using the SIMPOL programming language.

Conventions Used in This Book

Throughout the book, various conventions are used to identify items such as program code, file names, data types, etc. A `fixed-pitch` font is used for those sorts of things. Blocks of code are normally set off as a separate section, and have a gray background. Emphasized items are typically in an *italic* font.

Why SIMPOL?

The design ideas behind the SIMPOL programming language and the development tools associated with Superbase NG are a direct result of years of exposure to customer program code. We have looked at what worked well, at what caused problems, and have made many very strongly discussed decisions about fine points in the product. The goal for SIMPOL is to provide a powerful, yet easy-to-learn and use programming language. A common problem in the past has been that the programming languages that were easiest to start with did not enforce enough discipline to ensure that the products that often grew from those early simple programs were able to be maintained and extended. As a result, a powerful product might wither from a lack of resources to improve and support it, beyond a certain level of complexity. The design of SIMPOL promotes code reuse, careful design, and at the same time does not require an understanding of fairly arcane concepts in order to get started. It allows for both object-oriented and functional programming styles, and these can be mixed and matched where appropriate.

Another guiding principle for SIMPOL was that of being cross-platform. By designing the language to hide the vast majority of platform-specific issues, applications can be written once and run without change on other platforms.

Finally, in the heritage of numerous products from the mid-80's through the mid-90's, a conscious effort was made to provide higher level tools that assist people who may not have been trained programmers to succeed in solving their own problems. At the same time, we chose to build the tools in SIMPOL itself, so that they can be easily extended. By producing various layers of decreasing complexity within the same tool chain, people can enter into the development process at the level with which they are comfortable, and still grow and progress over time to more complex layers of the product if they choose. We found that although there are products that exist that allow easy development, they often become dead ends when the desires of the user exceed the capabilities of the tools. At the other end of the spectrum, there are many complex tools available on the market for doing development, but most are simply too difficult for people who are not programming for a living.

Running Superbase NG Programs

When Superbase NG is installed on Windows, an association is created between the `*.smp` file extension and the location of the `smprun32.exe` loader program. That means that any SIMPOL program can be run from the command line simply by typing its name, or by double-clicking it in Windows Explorer. If the program is a GUI-style program, then it will display a terminal window when the program is run. To avoid this, you can create a shortcut for the program that uses the `smpwin32.exe` loader program.

On Linux, the binary executables should normally be placed in the `/usr/bin/` directory, and the loadable libraries in the `/usr/lib/` directory. That means that programs can be run simply by using the `smprun` loader program without referring to its location. There is no special GUI loader program for Linux, as none is required. If desired, a shebang line can be added to the beginning of the `*.smp` file and the file made executable, and then the program can be run directly, as on Windows.

Deploying Superbase NG Programs

Once you have completed a program using Superbase NG and wish to distribute the results, you need to make sure that you include all the pieces necessary together with your compiled program. Since Superbase NG provides a component architecture, only the components required to run your application need to be redistributed with it. These files can be found in the `simpol\redist` directory. The essential components include the application loader, the core SIMPOL language library, and your program. In addition, if you used any components, then the associated library files must also be included. There are various loaders, depending on the type of program you are running. Below is a list of them:

Table 1.1. Superbase NG Runtime Loaders

Application Type	Loader for Win32	Loader for Linux x86
CGI – Web Server Applications	`smpcgi32.exe`	`smpcgi`
Fast-CGI – Web Server Applications	`smfcgi32.exe`	`smfcgi`
ISAPI – IIS Web Server Applications	`smisap32.dll`	N/A
Console Programs	`smprun32.exe`	`smprun`
GUI Programs	`smpwin32.exe`	`smprun`
Loader to call SIMPOL Functions as DLL Calls	`smexec32.dll`	N/A

There are two different loaders for regular programs on Windows; this is because Windows differs between programs that have their own window, and programs that do not. On Linux, all programs share the same loader program (except for specialized programs such as web server applications). The list of required libraries, arranged by component, is shown below:

Table 1.2. Superbase NG Runtime Components

Component	Required File(s) Win32	Required File(s) Linux x86
Web Server Applications	`smcgi32.dll`	`libsmpolcgi.so`
LXML – XML Document Object Model	`smlxml32.dll`, `iconv.dll`, `libxml2.dll`, `libxslt.dll`, `zlib1.dll`	`libsmpollxml.so`, Uses the libxml2 support from the distribution
ODBC – SIMPOL ODBC Client	`smodbc32.dll`	

Component	Required File(s) Win32	Required File(s) Linux x86
PPCS – SIMPOL Multi-User Database Client	`smppcs32.dll`	`libsmpolppcs.so`
PPSR – SIMPOL Multi-User Database Server	`smppsr32.dll`	`libsmpolppsr.so`
SBME – SIMPOL Single-User Database Client	`smsbme32.dll`	`libsmpolsbme.so`
SLIB – SIMPOL Shared Library Access (*.DLL, *.so)	`smslib32.dll`	`libsmpolslib.so`
SOCK – SIMPOL TCP/IP Socket Support	`smsock32.dll`	`libsmpolsock.so`
UTOS – SIMPOL File System Support	`smutos32.dll`	`libsmpolutos.so`
WXWN – SIMPOL GUI Components	`smwxwn32.dll`, `wxbase28u_vc_simpol.dll`, `wxmsw28u_adv_vc_simpol.dll`, `wxmsw28u_core_vc_simpol.dll`	`libsmpolwxwn.so`, `plus the wxWidgets runtime package for 2.8.x for the distribution`

Note

The only item listed above that cannot be distributed simply as a library file is the PPSR component. This is the code that implements the multi-user database server. To distribute the server, it is necessary to buy the appropriate database license and then use that registration number to install the Superbase NG database server engine on the customer's computer. Copying the PPSR component's DLL will result in the engine simply not working at all. It requires a correct installation with a valid registration number.

Summary

Since this is a quick start guide, as a starting point, it may be a good idea to take a quick look at what came in the package, and to divide it up into various areas.

Chapter 2. What's in the Package?

Overview of the Product

Superbase NG includes the following major sections:

- The Superbase NG IDE
- Superbase NG Personal
- The C-Language Components and Runtime Files
- The SIMPOL Language Libraries and Samples
- The Documentation
- General Utilities and Conversion Utilities for Legacy Superbase

Let's look at each of these items in more detail.

The Superbase NG IDE

The Superbase NG IDE is the place where program coding, compiling, and debugging are done. It is a carefully designed environment that closely supports the efforts of the programmer. For more information about this tool, it is recommended to read the first chapter of the SIMPOL IDE Quick Start Manual [http://www.simpol.com/docs/tutorial/], and for specifics about configuring the IDE see the Superbase NG IDE Users Guide [http://www.simpol.com/docs/ide/].

Superbase NG Personal

Superbase NG Personal is used for various things. It hosts the rapid application development (RAD) tools, such as the database table creation/modification tool, the display and print form designers, the graphic report designer, and a front-end for the reporting system. It also supplies a number of useful facilities, such as import/export, the ability to do basic data-entry into database tables (either through forms or in record view), reorganize utility, data update tool, labels system, etc.. In many cases, projects will begin in Superbase NG Personal and after the database tables, forms, and reports have been created, then the programmer will switch to building a basic program in the IDE to display the forms and to respond to events. Superbase NG Personal is not currently available as a separate package and it cannot currently be copied to other computers, it is part of the full development product.

C-Language Components and Runtime Files

SIMPOL is designed as a component-based architecture. Only the components required for a given program need to be distributed with that program. The pure minimum for a SIMPOL program is a loader and the core language library. The various components are listed below, including the Win32 and Linux filenames (when available):

- smpol — core language library — (`smpol32.dll`, `libsmpol.so`)
- sbsort — sorting orders for database components (ppcs, ppsr, sbme) — (`sbsort32.dll`, `libsbsort01.so`)
- CGI — CGI support — (`smcgi32.dll`, `libsmpolcgi.so`)
- ISAP — ISAPI support (Windows-only) — (`smisap32.dll`)

- LXML — XML and HTML DOM support — (`smlxml32.dll`, `libsmpollxml.so`)
- ODBC — ODBC client support (Windows-only) — (`smodbc32.dll`)
- PPCS — multi-user database access using the PPCS protocol — (`smppcs32.dll`, `libsmpolppcs.so`)
- PPSR — multi-user database server providing the PPCS protocol — (`smppsr32.dll`, `libsmpolppsr.so`)
- SBME — single-user database engine — (`smsbme32.dll`, `libsmpolsbme.so`)
- SLIB — shared-library function access (`*.dll`'s and `*.so`'s) — (`smslib32.dll`, `libsmpolslib.so`)
- SOCK — TCP/IP socket objects, client and server — (`smsock32.dll`, `libsmpolsock.so`)
- UTOS — utilities for file and operating systems — (`smutos32.dll`, `libsmpolutos.so`)
- WXWN — GUI objects and functions using wxWidgets — (`smwxwn32.dll`, `libsmpolwxwn.so`)

Of the items listed above, only the PPSR component requires a separate license (and installer) for distribution.

The following list contains the various loaders; again the file names are provided, first the Win32 file name and then the Linux file name. The debugging versions of the loaders are not intended for redistribution:

- smexec — the DLL loader program for loading SIMPOL language libraries from other programming languages — (`smexec32.dll`)
- smpcgi — the CGI loader program — (`smpcgi32.exe`, `smpcgi`)
- smgd1_ — the debug CGI loader program — (`smgd1_32.exe`, `smgd1_`)
- smfcgi — the Fast-CGI loader program — (`smfcgi32.exe`, `smfcgi`)
- fcgd1_ — the debug Fast-CGI loader program — (`fcgd1_32.exe`, `fcgd1_`)
- smprun — the console loader program — (`smprun32.exe`, `smprun`)
- smpd1_ — the debug console loader program — (`smpd1_32.exe`, `smpd1_`)
- smpwin — the Windows GUI loader program (not needed by Linux) — (`smpwin32.exe`)
- smpw1_ — the debug Windows GUI loader program — (`smpw1_32.exe`)

SIMPOL Language Libraries and Samples

In keeping with our company philosophy of making sure that we use our own products (to keep us in touch with the needs of the customers), many of the lower level and higher level objects and functions are written in the SIMPOL programming language. Many of these are provided as full source code with the entire project code as samples.

The SIMPOL language components are found in the `\lib` directory for inclusion in projects. Most of them have equivalent SIMPOL language projects in the `\projects` directory. Programming samples are also primarily found in the `\projects` directory. In the `\samples` directory there are three subdirectories, each of which contains SIMPOL source files (but not projects), except for one that contains SBL programs. The SBL programs demonstrate methods of calling SIMPOL from SBL, one of which determines the dimensions of a JPEG image, the other makes the operating system "File Open" and "File Save" dialogs usable from an SBL application. There are also a group of bitmap resources in the `\resources` directory. Finally, in the `\include` directory are includable SIMPOL source files for defining useful constants, such as standard error values, or for working with specific libraries. The list of standard libraries continues to grow. In the following sections we will look at what is provided in more detail.

SIMPOL Language Libraries (`*.sml`)

- `abs.sml` — Implements the `ABS()` function for returning the absolute value of a number or integer
- `appframework.sml` — Implements a fairly powerful application framework for creating GUI-style database-based applications
- `boolstr.sml` — This provides functions for converting boolean and datetime values into strings and also the reverse
- `bzip2.sml` — This provides functions that wrap the `BZip2.dll` compression library
- `calceval.sml` — Provides a function that can evaluate a string containing a formula and return the result
- `calclib.sml` — Provides a basic calculator that can be popped up and which returns the final result
- `codepageslib.sml` — Code page conversion library for converting to and from various code pages
- `colorpalette.sml` — Provides a blob based storage for an image palette
- `commonreportgui.sml` — Library containing the types and functions used to produce elements for the report system, such as the filter dialog, calculation dialog, and the sort order dialog
- `conflib.sml` — Library of functions for reading from and writing to configuration files in the Microsoft INI file format
- `consolelib.sml` — Library containing the tConsole type for creating console-style progams that allow interaction with the user.
- `databaseforms.sml` — Data-aware form library providing the primary interface for working with data-aware forms and form controls
- `datetimelib.sml` — Collection of functions and types used to provide conversions from and to dates, times, and datetimes, in various formats
- `db1lib.sml` — Basic stub library that should never be called directly but which acts as a supplier of information to the IDE during development when using the `type(db1table)` family of type tags
- `db1util.sml` — Database utilities library with routines for copying records, creating and maintaining system tables, etc.
- `dbconverter.sml` — Import/Export conversion library with support for ASCII-Delimited, CSV, XML, SBM, and PPCS
- `displayformat.sml` — Dialogs for retrieving the desired display format for various data types
- `drilldown.sml` — Dialog for allowing interactive search against an index in a database table with display of requested columns and return of selected record

- `dxflib.sml` — Function to convert AutoCAD DXF files of a specific style into Windows bitmaps
- `errormsgs_en.sml` — Library that converts an error code into a an English-language message that describes the error
- `fastset.sml` — A set object that allows elements to be string indexed objects of any type, implemented using red/black trees
- `filesyslib.sml` — Functions for working with elements of a file system, such as parsing pathnames, retrieving the current directory, etc.
- `filtergui.sml` — Provides the selection filter GUI functionality
- `formlib.sml` — Functions and types for loading and saving data-aware forms and for saving forms as source code
- `gaugelib.sml` — Provides various progress gauge dialog types
- `graphicreportlib.sml` — Provides types and functions that implement a banded report writer for use with SIMPOL databases for output to window or printer
- `httpclientlib.sml` — Objects for retrieving items from the web using `GET` or `POST`
- `ieeelib.sml` — Contains function for converting to and from 4-byte and 8-byte IEEE floating point format
- `imagelib.sml` — Functions and types for reading and writing images in BMP and XPM format
- `int.sml` — Implements the `INT()` function for converting a number to an integer
- `jpeglib.sml` — Currently only supplies functions for determining the dimensions of an image in JPEG format
- `json.sml` — Provides functions and types to work with JSON-encoded data, including converting to and from SIMPOL
- `labelslib.sml` — Implements a mailing labels package including defining, saving, loading, and printing of labels
- `libxml.sml` — Implements the Document Object Model Core Level 1 and 2 and part of 3 plus XSLT transforms and XPATH by working with the LXML component
- `lists.sml` — Utility library providing various strutural types, such as nodes, lists, rings, queues, and stacks
- `logmanager.sml` — Utility library providing a mechanism to allow multiple threads to write to a text-based log file by providing a queuing system and a function to process the queue
- `ltrim.sml` — Implements the `LTRIM()` function to trim spaces from the left of a string
- `mathlib.sml` — Contains various math functions such as `pi()`, `sqrt()`, `sin()`, `cos()`, `tan()`, and more
- `mrulib.sml` — Provides a library of types and functions for working with most-recently-used lists, including loading and saving to INI files and managing a submenu
- `netinfolib.sml` — Contains functions like `getusername()` and `getcomputername_win32()`
- `objset.sml` — An early implementation of a set object based on binary trees, but new code should use the `fastset.sml` library or the internal set type

- `odbc2.sml` — Helper library for working with the ODBC client functionality that is part of the SIMPOL ODBC component
- `pad.sml` — Implements the `PAD()` function to right-fill a string with spaces to a specified size (or to truncate if it exceeds that size)
- `parsenum.sml` — Is a contribution from a member of the SIMPOL community and provides a function that converts numbers into words (in English), commonly used in check writing programs
- `printformlib.sml` — Contains useful functions for printing to window or printer, like `printwxform()`, `printrecord()`, and `printtext()`
- `propertybrowser.sml` — This implements a runtime property browser that can be very useful in tracking down the value of objects at runtime (supervisor functionality)
- `q1.sml` — This is the library that implements the query optimizer for the SIMPOL report engine
- `quickreportlib.sml` — Contains types and functions that implement the light-weight reporting functionality called Quick Report that can output to window, printer, database, clipboard, and other targets and which can create, save, and load the reports
- `random.sml` — Provides the random type for use in generating pseudo-random numbers
- `recordview.sml` — Provides a record view implementation used by the application framework
- `registrylib.sml` — Contains the win32registry type that provides methods for reading and writing the Windows registry
- `reorglib.sml` — Functions for reorganizing (repacking) databases in the SIMPOL database engine format (`*.sbm`)
- `repguilib.sml` — This provides the Quick Report front-end for user programs
- `replace.sml` — Provides the `replace()` function for replacing all instances of one substring with another in a target string
- `reportlib.sml` — Contains the types and functions that implement a base reporting system used by more sophisticated wrappers such as Graphic Report and Quick Report
- `rsalib.sml` — Contains functions and types for working with RSA encryption, including key generation, encryption and decryption
- `sbislib.sml` — Functions for working in a CGI environment, such as `HtmlInclude()`, or `HtmlRead()`
- `sbldatelib.sml` — Implements functions for working with dates and for formatting dates as strings
- `sblexten.sml` — Includes a conversion of functions from a Superbase sample library of the same name and which may be helpful when converting from Superbase
- `sbllib.sml` — Functions are provided that represent various FN-style functions from SBL, such as `FN_Dec()`, `FN_Fact()`, etc.
- `sbllocaledateinfo.sml` — Contains the SBLlocaledateinfo type for use with the date formatting functions in `sbldatelib.sml`
- `sbltimelib.sml` — Functions for formatting times as strings and converting strings back to time values
- `sbnglib.sml` — Contains useful types and functions for interfacing with Superbase NG Personal, such as rings of data sources and tables, and an object for managing wxformoption objects

- `sendkeys.sml` — This library implements a SENDKEYS functionality for Win32
- `sendmail.sml` — Provides the `sendmail()` easy wrapper function to the `smtpclientlib.sml` functionality for sending text-based SMTP emails
- `serialize.sml` — This library implements a serialization mechanism for storing objects at runtime and reloading them later
- `sessionid.sml` — Functions and types for creating and manipulating session IDs using cookies for web applications
- `sessionid2.sml` — Functions and types for creating and manipulating session IDs without cookies for web applications
- `shellexecute.sml` — Wrapper around the Windows API call for loading the appropriate executable for a given file type, ie. Acrobat Reader for `*.pdf` files
- `simpollib.sml` — Contains functions that use meta-capabilities of SIMPOL to provide functions like: `findfunction()` and `isproperty()`
- `smtpclientlib.sml` — Email functionality via SMTP
- `smtpdatelib.sml` — Implementation of a date formatting function that accepts format strings using the standard SMTP date format
- `sortlib.sml` — Various sorting algorithms such as Insertion sort, Quick Sort (iterative and recursive), etc.
- `soundlib.sml` — Provides sound playback functionality that is currently Windows only
- `sql1.sml` — The library providing a SQL92 report engine for SIMPOL databases
- `str.sml` — Provides the `STR()` for formatting a numeric value as a string using a pattern
- `stringlib.sml` — Numerous functions that implement useful string handling functionality, including `parsetoken()`, `ltrim()`, `rtrim()`, etc.
- `tableview.sml` — Provides a table view implementation used by the application framework
- `timer.sml` — Provides a timer type that can call an event handler either once or at intervals and which runs in a separate thread
- `trim.sml` — Contains the `TRIM()` function
- `uisyshelp.sml` — Various functions and types for providing standard system defaults, such as system colors, default fonts, display size, etc.
- `unittest.sml` — Basic unit testing library that helps in running regression tests
- `urlendecode.sml` — Functions for doing URL-encoding and URL-decoding
- `urllib.sml` — Provides the URL type and the `parseurl()` function for parsing a URL into its component parts
- `utf8lib.sml` — Functions for converting from and to UTF-8 format
- `uuencode.sml` — Functions for doing uuencode, uudecode, base 64 encoding and decoding and quoted printable encoding
- `val.sml` — Implements the `VAL()` function
- `volatable.sml` — Provides a fairly full implementation of a database that only exists in memory. Compatible to the sbme1 family except for table modification

- `windowsemaillib.sml` — Contains a data type for sending email by using the Windows scripting host
- `winfiledlg.sml` — Provides a wrapper to the open and save dialogs from the operating system that can be called via the `smexec32.dll` from a program such as Superbase
- `xmllib.sml` — Provides a number of useful programs for parsing and evaluating XML strings and can be used to enhance the functionality provided by `libxml.sml`

Supplied Superbase NG Projects

Superbase NG ships with a large number of sample projects. Many if not most of the libraries listed above are included as source code projects. Below is a basic description of the directories containing projects:

Console Projects

The `console` directory houses projects that demonstrate basic functionality and are meant to be run from the console:

- `convert` — Demonstrates a command line program for converting end of line characters in text files from DOS (CRLF) to Linux (LF) annd also to the older Macintosh format (CR)
- `hello` — The usual "Hello World" program
- `ppcsselectkey` — A command line program for selecting a specific record from a table on a PPCS server and then showing the content of that record.
- `urldump` — A command line program for retrieving a page from the World Wide Web and either storing it in a file or dumping it to the console.

XML Document Object Model (DOM)

The `DOM` directory contains a sample program that fully exercises the XML DOM

- `libxml_example` — Contains 14 tests that demonstrate the various features of the XML DOM support in SIMPOL

Examples

The `examples` directory contains GUI programs, TCP/IP sockets programs, and samples of using the `lists.sml` library. To begin with, the sockets examples are a pair of projects. The `client` project and the `server` project are designed to work together to demonstrate transferring a file from a server to a client upon request of the client.

- `client` — Demonstrates using a TCP/IP-based client program to connect to a server and to communicate with it, including receiving both text and binary data.
- `server` — Shows how to create a custom TCP/IP-based server program that implements a basic protocol and which then acts upon connections and can send and receive text data and which then sends binary data.

There are four samples that demonstrate the use of various types from the `lists.sml` library. These are:

- `listsample` — This example demonstrates how to use the list and listnode types in order to wrap existing objects and possibly add information to them in addition to managing them in a list.
- `ringsample` — This example demonstrates how to use the ring and listnode types in order to wrap existing objects and possibly add information to them in addition to managing them in a list.

- `dlistsample` — This example demonstrates how to use the dlist and dlistnode types in order to wrap existing objects and possibly add information to them in addition to managing them in a list. Both dlist and dring types are better choices when you need to insert or delete from the list or ring, since they are more efficient in these operations.

- `dringsample` — This example demonstrates how to use the dring and dlistnode types in order to wrap existing objects and possibly add information to them in addition to managing them in a list. Both dlist and dring types are better choices when you need to insert or delete from the list or ring, since they are more efficient in these operations.

There is a single example that shows the use of the dataform1 type family and which loads any valid form stored as an `*.sxf` file (opening the data sources that are referenced in the form) and then allows the browsing of records using the form. That example is:

- `dataforms` — Loads and allows the browsing of data in any valid form

The `udtmemberopsample` project explains the implementation and use of the SIMPOL member operator (!) in a user-defined type. It is an advanced topic, but can be useful depending on the complexity of the application being developed.

- `udtmemberopsample` — demonstrates the use of the SIMPOL member operator in a user-defined type

The remaining projects in this directory all demonstrate various capabilities within the GUI controls provided via the wxWidgets library.

- `ole2excel` — is a sample program that demonstrates the use of OLE2 automation to open Excel, create an Excel workbook, add data, select and calculate that data, create a chart, and then read the results back out into SIMPOL

- `wxdialog` — is a very small program that demonstrates the use of the wxdialog type to create a modal dialog and wait until it is closed or the OK is pressed.?

- `wxdialog2` — is a very small program that demonstrates the use of the wxdialog type to create a non-modal dialog and wait until it is closed or the OK is pressed.

- `wxdialog3` — is a very small program that demonstrates the use of the wxdialog type to create a modal dialog using the standard buttons feature, and then waits until it is closed or the OK is pressed.

- `wxdialog4` — is a very small program that demonstrates the use of the wxdialog type to create a non-modal dialog using the standard buttons feature, and then waits until it is closed or the OK is pressed.

- `wxforms` — is a minimal program that demonstrates the use of the wxform and wxwindow types to display a form in a window and then wait for events.

- `wxforms2` — is a small program that demonstrates the use of the wxform and wxwindow types together with a group of form controls on the form that allow modification and include sample data. Pressing the button will evaluate the selections and content from the various form controls, close the window and return that as a result.

- `wxgrid` — is a small program that demonstrates the use of the wxgrid including various aspects of using the grid control.

- `wxmenu` — is a minimal program that demonstrates the use of the wxmenubar, wxmenu, and wxmenuitem types to create a basic menu bar that shows the various features supported and to add that to a window.

- `wxwindows` — demonstrates the minimal amount required to create a window on the screen and then wait for events.

- `wxwindows2` — creates four different windows of various styles. Closing any of the windows closes all of them and ends the program.

Forms Examples

The `forms` directory contains some more sophisticated form-based GUI examples. One of them shows various controls in a number of configurations, the other is a dedicated import program that can import from PPCS data sources into SBME. It was originally designed to assist the conversion of applications from Superbase into SIMPOL.

- `demo` — demonstrates the use of the various features that SIMPOL provides via wxWidgets.
- `importppcs2sbme` — implements an import program that makes use of PPCS, SBME, plus the GUI components via wxWidgets.

Games

The `games` directory is meant to contain example game programs. The first one included shows the object-oriented implementation of the classic worm game.

- `worm` — Uses a minimal set of libraries to produce a basic version of the classic worm game.

Libraries

The largest number of samples can be found in the `Libs` directory, which contains projects that implement reusable functionality either as functions or types.

- `ABS` — implements the `ABS()` function for returning the absolute value of a number or integer
- `appframework` — provides a fairly powerful application framework for creating GUI-style database-based applications
- `boolstr` — formatting library for converting boolean and datetime values to string and back
- `bzip2` — compression library wrapper for the BZip2 compression format and the `BZip2.dll`
- `calceval` — library for evaluating a string and parsing and carrying out the calculation formula and returning the result
- `calclib` — library that displays a calculator and that returns the result of the calculation
- `codepageslib` — code page conversion library for converting to and from various code pages
- `conflib` — library of functions for reading from and writing to configuration files in the Microsoft INI file format
- `consolelib` — library that implements a basic console window for creating console programs that interact with the user
- `databaseforms` — library of types and functions that provide the full implementation of data-aware display and print forms for SIMPOL
- `datetimelib` — library of functions for converting to and from dates, times, and datetimes (includes various other libraries plus its own functions)
- `db1lib` — library that implements a stub class to match the db1 type tag family so that the IDE will provide useful information when working with variables that are defined as `type(db1table)` for example
- `db1util` — database utilities library with routines for copying records, creating and maintaining system tables, etc.

- `drilldown` — user-interface component that provides an interactive search capability with a display of results in a grid control and return of the selected record
- `dxflib` — function for converting a specific style of `*.dxf` file into a Windows bitmap (includes a helper DLL)
- `fastset` — a set implementation compatible with the objset type but faster using red-black trees
- `filesyslib` — functions for working with elements of a file system, such as parsing pathnames, retrieving the current directory, etc.
- `formlib` — types and functions for loading, saving and saving as source code dataform1 and printform1 objects
- `gaugelib` — provides various progress gauge dialog types
- `httpclientlib` — objects for retrieving items from the web using `GET` or `POST`
- `imagelib` — functions and types for reading and writing images in BMP and XPM format
- `INT` — implements the `INT()` function for converting a number to an integer
- `iplib` — library for hosting functions and types associated with working with the Internet protocol
- `jpeglib` — currently only supplies functions for determining the dimensions of an image in JPEG format
- `lists` — utility library providing various structural types, such as nodes, lists, rings, queues, and stacks
- `LTRIM` — implements the `LTRIM()` function to trim spaces from the left of a string
- `mathlib` — contains various math functions such as `pi()`, `sqrt()`, `sin()`, `cos()`, `tan()`, and more
- `mrulib` — provides a library of types and functions for working with most-recently-used lists, including loading and saving to INI files and managing a submenu
- `netinfolib` — contains functions like `getusername()` and `getcomputername_win32()`
- `objset` — an early implementation of a set object based on binary trees, but new code should use the `fastset.sml` library or the internal set type
- `PAD` — implements the `PAD()` function to right-fill a string with spaces to a specified size (or to truncate if it exceeds that size)
- `parsenum` — is a contribution from a member of the SIMPOL community and provides a function that converts numbers into words (in English), commonly used in check writing programs
- `printformlib` — contains useful functions for printing to window or printer, like `printwxform()`, `printrecord()`, and `printtext()`
- `random` — provides the random type for use in generating pseudo-random numbers
- `registrylib` — contains functions for working with the Windows registry
- `replace` — provides the `replace()` function for replacing all instances of one substring with another in a target string
- `rsalib` — contains functions and types for working with RSA encryption, including key generation, encryption and decryption

- `sbislib` — functions for working in a CGI environment, such as `HtmlInclude()`, or `Html-Read()`
- `SBLDateLib` — implements functions for working with dates and for formatting dates as strings
- `sblexten` — includes a conversion of functions from a Superbase sample library of the same name and which may be helpful when converting from Superbase
- `sbllib` — functions are provided that represent various FN-style functions from SBL, such as `FN_Dec()`, `FN_Fact()`, etc.
- `SBLlocaledateinfo` — contains the SBLlocaledateinfo type for use with the date formatting functions in `sbldatelib.sml`
- `SBLTimeLib` — functions for formatting times as strings and converting strings back to time values
- `sbnglib` — contains important types used throughout much of SIMPOL, such as datasourceinfo and tbinfo as well as the types and functions used to provide option groups for wxformoption types
- `sendmail` — provides the `sendmail()` easy wrapper function to the `smtpclientlib.sml` functionality for sending text-based SMTP emails
- `serialize` — provides the ability to serialize an object to a file and then read the data from the file and recreate the object at a later point in time
- `shellexecute` — wrapper around the Windows API call for loading the appropriate executable for a given file type, ie. Acrobat Reader for `*.pdf` files
- `simpollib` — contains functions that use meta-capabilities of SIMPOL to provide functions like: `findfunction()` and `isproperty()`
- `smtpclientlib` — email functionality via SMTP
- `smtpdatelib` — implementation of a date formatting function that accepts format strings using the standard SMTP date format
- `sortlib` — various sorting algorithms such as Insertion sort, Quick Sort (iterative and recursive), etc.
- `soundlib` — Library with the long-term plan to be the host for sound playback routines, currently supports Windows sound playback
- `STR` — provides the `STR()` for formatting a numeric value as a string using a pattern
- `stringlib` — numerous functions that implement useful string handling functionality, including `parsetoken()`, `ltrim()`, `rtrim()`, etc.
- `timer` — provides a timer type that can call an event handler either once or at intervals and which runs in a separate thread
- `TRIM` — contains the `TRIM()` function
- `uisyshelp` — various functions and types for providing standard system defaults, such as system colors, default fonts, display size, etc.
- `unittest` — basic unit testing library that helps in running regression tests
- `urlendecode` — functions for doing URL-encoding and URL-decoding
- `urllib` — split out of a small library containing a type and function for parsing a URL from a string into its component parts

- `utf8lib` — functions for converting from and to UTF-8 format
- `uuencode` — functions for doing uuencode, uudecode, base 64 encoding and decoding and quoted printable encoding
- `VAL` — implements the `VAL()` function
- `windowsemaillib` — data type for sending email by using the Windows scripting host
- `winfiledlg` — provides a wrapper to the open and save dialogs from the operating system that can be called via the `smexec32.dll` from a program such as Superbase
- `xmllib` — provides a number of useful programs for parsing and evaluating XML strings and can be used to enhance the functionality provided by `libxml.sml`

SBME Database Examples

A series of command line programs that make use of the single-user database engine can be found in the `sbme` directory. This includes two utility programs for doing database maintenance and repair.

- `jdktutorial` — was inspired by one of our users. It provides a basic tutorial on using the sbme1 family of types to first create, then populate a table in a container file. Then to open lock and modify records in that container file.
- `reorganize` — provides a command line front-end to the functionality in `reorglib.sml` for repacking one or more database tables.
- `sbmecust` — reads records from the CUST table at simpol.com port 1280 and creates an `*.sbm` file with the same table.
- `sbmecust2` — provides a timing test for reading records from one table and creating with them another table.
- `sbmecust3` — demonstrates creating a duplicate table in the the same container as the original table.
- `sbmereadcust` — shows how to read records from a table and output them to a text file in XML format.
- `sbmerepair` — provides a command line based repair program to fix a database table in the unlikely event that it may have become corrupted.

SIMPOL Tutorial Examples

All of the sample programs used in this book that are not located elsewhere can be found in the `Projects\tutorial` directory.

- `addressbook` — implements a basic address book application using data-aware forms and the application framework.
- `colorlab` — demonstrates a dialog-style application program.
- `quickreportsample` — shows how to create a Quick Report program in source code.
- `sbair` — contains the form, program, and required database tables from the Superbase Airlines sample that is converted in the Chapter 11, *Converting Legacy Superbase* chapter.
- `simpolbusiness` — contains the forms, reports, program, and required database tables for the SIMPOL Business sample from the Chapter 7, *SIMPOL Business* chapter.

SIMPOL Web Server Programs

A full suite of web server programs, which work using CGI or ISAPI are contained in the `ssp` directory. To use the samples, make sure that the items from the `Apache` directory are placed in the appropriate locations. The items from the `htdocs` directory include the items in the `css` directory and the images that are located in the `images` directory. The items in the `cgi-bin` should be placed in the appropriate cgi-bin location in your system. These items are used by the web server sample projects. There is also a directory called: `include`, which contains chunks of code used by more than one project in this group. The `ppcsserver` directory contains a Superbase server program and associated database files, as well as a SIMPOL server program (`simpolserver.smp`) together with the necessary database container files and the configuration file for the server (`sspsamples.cfg` — read the `readme.txt` to run or stop the server).

The examples in this section do not demonstrate particularly attractive web pages. They are very basic in their look and feel. Designing attractive web pages is better done in an HTML authoring tool. These examples demonstrate how to use SIMPOL to dynamically create pages, as well as showing how to interact with the database and to accept data over the web. Most of the more interesting SIMPOL-based web applications are normally designed using CSS and XHTML in a proper authoring tool, and then the template pages are migrated into SIMPOL and set up to use parameters so that they can be output with varying content by the programs.

Note

All of the samples that begin with "sbis" were translated from the original samples included with the Superbase Internet Server program (SBIS).

- `hellocgi` — is designed using the "server page" approach. That uses a file called `hellocgi.smz`, which is an HTML file containing special comments that are processed by the SIMPOL IDE into a SIMPOL program file with the same name. This is then compiled as part (or in this case as all) of the program.
- `sbiscalendar` — demonstrates creating and outputting a calendar using a table.
- `sbiscontact` — is a sample contact database with the ability to view the records in a simple record view format. It also allows browsing with First, Previous, Next, and Last buttons. These all call the same program with specific parameters. On the right-hand side are another set of buttons: Add, Search, Report, and Report2. These buttons call either directly or indirectly other programs in the group.
- `sbiscontactdisplay` — displays a selected record (found as a result of searching using the Search button. If the selected record is not found, it redisplays the search page.
- `sbiscontactpost` — handles the posting of a new record as a result of a cal to the Add function in `sbiscontact`.
- `sbiscontentsframe` — is part of the `sbisframesample`, although frames have now gone very out of fashion in web design.
- `sbisenvvars` — outputs all the various CGI variable values (plus those supplied by ISAPI if called from IIS) that can be retrieved via the cgicall type's `getvariable()` method.
- `sbisframesample` — uses various SIMPOL programs to provide different frame content for each of the various frames. Frames are no longer particularly popular in modern web design, but can still have their uses.
- `sbisimagesample` — demonstrates a dynamic page that shows an image.
- `sbisincludesample` — shows the method of including an external HTML file into the output going back from the program to the browser.

- `sbismainframe` — is part of the `sbisframesample`, although frames have now gone very out of fashion in web design.
- `sbisreport` — runs an unoptimized report that shows its results in a table and does a three-level sort of the results. Each line of the results can be clicked on to call the `sbiscontactdisplay` code to show the record for the resulting selection.
- `sbisreportfast` — runs an optimized report that shows its results in a table and does a three-level sort of the results. Each line of the results can be clicked on to call the `sbiscontactdisplay` code to show the record for the resulting selection.
- `sbistitleframe` — is part of the `sbisframesample`, although frames have now gone very out of fashion in web design.

Tests

A few basic test programs can be found in the `tests` directory.

- `calcevaltest` — tests the various capabilities of the `calceval()` function using the `unittest.sml` library.
- `chartest` — could actually be called echo, since it returns whatever text is passed in the first parameter to the function.
- `consoletest` — is a simple demonstration program that shows how to use the `consolelib.sml` library to create a simple tet-based program that can interact with the user.
- `datelibtest` — is a test suite using the `unittest.sml` to implement regression testing for the `DATESTR()` and `string2date()` functions.
- `fcasetest` — is a test suite using the `unittest.sml` to implement regression testing for the `fcase()` function.
- `filetypetest` — is a test suite using the `unittest.sml` to implement regression testing for the `filetype()` function.
- `fixtest` — is a test suite using the `unittest.sml` to implement regression testing for the `.fix()` function.
- `string2valtest` — is a test suite using the `unittest.sml` to implement regression testing for the `string2val()` function.
- `STRtest` — is a test suite using the `unittest.sml` to implement regression testing for the `STR()` function.
- `timelibtest` — is a test suite using the `unittest.sml` to implement regression testing for the `TIMESTR()`, `extTIMESTR()`, and `string2time()` functions.

Documentation

Superbase NG contains six different books that cover various aspects of using the product. These are:

- SIMPOL Quick Start Guide (this book)
- *Superbase NG IDE Users Guide* [http://www.simpol.com/docs/ide/]
- *SIMPOL Language Reference Manual* [http://www.simpol.com/docs/langref/]
- *SIMPOL Programmer's Guide* [http://www.simpol.com/docs/progbook/]
- *Superbase NG IDE Quick Start Manual* [http://www.simpol.com/docs/tutorial/]

- *Superbase NG Personal User Guide* [http://www.simpol.com/docs/personal/]

Utilities

There are a number of standalone utility programs included with Superbase NG. These include:

- `imagetool.smp` — provides a tool for manipulating images for creating bitmap buttons (contributed by John Roberts)
- `projectfixer.smp` — a program for changing the paths in one or more project files in a subdirectory tree
- `saveimagetoblobgui.smp` — a program that converts one or more images and writes out a SIMPOL source code program that for each image provides a function that contains the image as a blob and can return it as a wxbitmap
- `sbf2sbm.smp` — a program for converting a list of Superbase database files into SIMPOL database files
- `sbm2smagui.smp` — a program for converting one or more Superbase NG database tables into a function that can re-create the table (empty of data), which is useful for generating tables at run time
- `sbmerepair.smp` — a program for repairing a database by retrieving all records via their unique internal record ID (ignores indexes completely and can also ignore sequential linking problems)
- `sbv2sxf.smp` — a program for converting a list of Superbase form files into Superbase NG display form files

Superbase Conversion Utilities

To support Superbase programmers and users, there are a number of conversion tools that are provided in the Superbase SBL programming language. These are:

- `dlg2sma.sbp` — a converter for Superbase dialog programs saved from the Superbase Dialog Editor into SIMPOL source code to create the equivalent form using the wxWidgets-based components
- `ngmengen.sbp` — converts Superbase menu programs saved from the Superbase Menu Editor into SIMPOL source code using the wxWidgets-based components
- `sbv2sxp.sbp` — generates a SIMPOL XML print form file that is directly loadable from the SIMPOL data-aware print form support, assuming that the database tables have been saved as SIMPOL equivalents, which can be used as the starting point for working with the form in the SIMPOL Print Form Designer
- `sbvr2xml.sbp` — generates an imperfect SIMPOL XML graphic report file that requires some modification under certain circumstances but should then be directly loadable from the SIMPOL graphic report code, assuming that the database tables have been saved as SIMPOL equivalents, which can be used in SIMPOL programs as loadable and runnable Graphic Reports

Chapter 3. Getting Started

The Essentials

It must have been clear by then end of the previous chapter, that there are a lot of different ways to approach Superbase NG. The hardest part about getting to know a new product is that there are so many things to learn. It is generally best to have some clear goals, in order to direct the learning and to provide an early project or two. As a starting point, it is strongly recommended that the reader at least browse the first chapter of this book. Also if any real programming is planned it is a good idea to work through Chapter 1 of SIMPOL IDE Quick Start Manual [http://www.simpol.com/docs/tutorial/], which teaches the basics of using the IDE to create projects, edit, compile, and debug programs, to work with external libraries, and to set project settings.

Once that is done, the next steps depend greatly on what the reader wishes to accomplish. These might be any of the following:

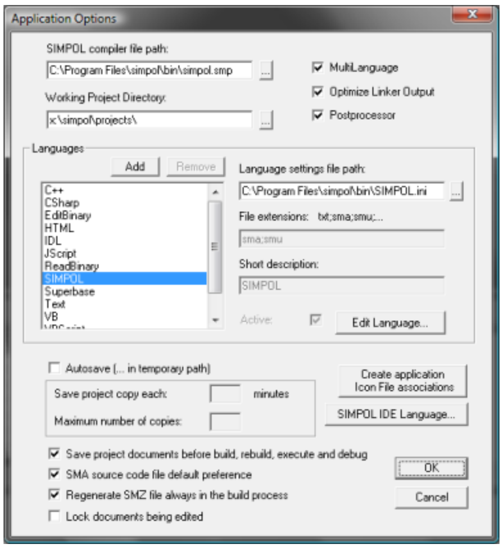
- A command line program, see the section called “Command Line Programs”
- A dialog-style application, see the section called “Dialog-Style Programs”
- A GUI-style database program, see the section called “Database GUI Applications”
- A web server program, see the section called “Web Server Applications”
- A standalone server application, see the section called “Server Applications”
- A conversion from Superbase, see the section called “Converting from Superbase”

Each of these is described more thoroughly below. Obviously these are merely starting points, there is nothing that says that they couldn't be combined in various ways, such as a server application that has a GUI for controlling it, or database GUI application that also provides a set of web server applications to allow some users a specific set of functionality via the web to what is otherwise a desktop application. Another might be a desktop application that uses the `httpclientlib.sml` library to access useful resources on the Internet and provide their functionality to the desktop program.

Preparing Our Environment

Before we start actually developing any programs, it might be a good idea to start out by preparing our development environment. There isn't much to do, and it isn't absolutely essential, but it will save time and aggravation later, especially if you are using the Windows Vista operating system or a later version. That is because it is very difficult, bordering on impossible to manage projects as a sub-directory below the `Program Files` directory. There is a special level of additional protection that prevents applications from writing to that directory, even if you have administrative rights. As such, it is a good idea to get in the habit of locating your projects somewhere else, such as your home directory. In Windows XP and earlier that is normally the `C:\Documents and Settings\username\My Documents` directory. In Vista, it has been changed to `C:\Users\username` and even more importantly, the actual home directory is more usable than it was in XP.

In the SIMPOL IDE select Tools → Options.... In that dialog window, in the edit control for the Working Project Directory, enter a path name or click on the … next to the field and select the path from the directory selection tool. The dialog window can be seen here:


Application Options

SIMPOL compiler file path: C:\Program Files\simpol\bin\simpol.smp

Working Project Directory: x:\simpol\projects\

☑ MultiLanguage
☑ Optimize Linker Output
☑ Postprocessor

Languages

C++
CSharp
EditBinary
HTML
IDL
JScript
ReadBinary
SIMPOL
Superbase
Text
VB

Language settings file path: C:\Program Files\simpol\bin\SIMPOL.ini

File extensions: txt;sma;smu;...

sma;smu

Short description: SIMPOL

Active: ☑

☐ Autosave (... in temporary path)

Save project copy each: minutes

Maximum number of copies:

☑ Save project documents before build, rebuild, execute and debug
☑ SMA source code file default preference
☑ Regenerate SMZ file always in the build process
☐ Lock documents being edited

The IDE Application Options dialog.

Command Line Programs

Getting started with command line programs is probably as easy as it gets in SIMPOL, from a purely programming perspective. There are a number of examples to show the ropes. Command line programs can take up to 10 parameters (currently), and can output their results. They do not have access to typical command line features like stdin, stdout, and stderr, but can still accomplish goals and return results. For an in-depth look at creating a command line program, see Chapter 4, *Command Line Programs*.

Dialog-Style Programs

These types of programs are normally not terribly complicated, and are often designed to provide a tool that accomplishes a specific goal. The applications are usually hosted in a dialog window and are generally not connected with a database (though they certainly could be). For the complete story, with a working example, visit Chapter 5, *Dialog-Style Programs*.

Database GUI Applications

Database programs in SIMPOL tend to start with Superbase NG Personal. Using its table creation tool, or via the import functionality, a new database table or tables can be created. In the same program the Form Designer can be found. Once the database tables are created, the Form Designer is used to create appropriate masks for the screen. These can be saved as forms, as source code, or both. Once the basic components have been created, they can be quickly turned into a small program that provides all the tools for creating, editing, and deleting data from the tables. More can be added to allow the output of data in various formats. For a full example, see Chapter 6, *GUI-Style Database Programs*.

Web Server Applications

SIMPOL can also be used to create powerful web server applications and has built-in support for CGI (Common Gateway Interface), ISAPI (Internet Server Application Programming Interface), and Fast-CGI, which is a high-performance version of CGI. One of the more powerful features in SIMPOL when developing web server applications is the ability to do source-level debugging of a web server application as a callback from the web server. Generally this is done using the Apache web server running locally on Windows. For the complete story, go to Chapter 9, *Web Server Programs*.

Server Applications

Server applications are a special type of command line program. They are designed to start up and then wait for clients to connect to them. At that point, they provide some service. Typically they are using TCP/IP to communicate, though a SIMPOL database server program is also a server program that is waiting for connections from database clients. Any sort of service could be a viable candidate, such as a program that does some very complex calculations based on a specific set of input, or a program that regularly collects information from various web sites, consolidates that, and produces a new set of information based on what it found and makes that available to clients. Another example might be a dedicated encryption/decryption service, for communications security. To have a look at creating a server program, see Chapter 10, *Server Programs*.

Converting from Superbase

Superbase provided a great set of tools to quickly create applications, similar in style to the database applications described above. Superbase is not directly compatible with SIMPOL, but a significant effort has been made to ease the path of migration from Superbase to SIMPOL. To that end a number of conversion tools are included, some in SIMPOL and some in SBL. For a walk through the process of doing a Superbase to SIMPOL migration, Chapter 11, *Converting Legacy Superbase* is the best place to start.

Chapter 4. Command Line Programs

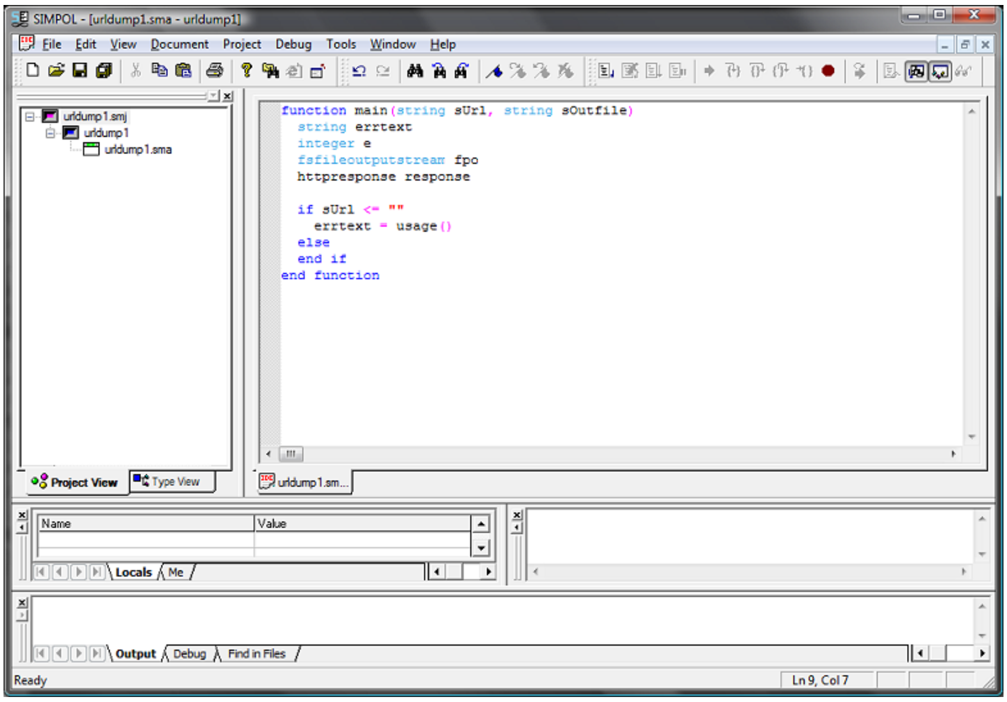
Building a Command Line Program

In this chapter we are going to build a program that based on the parameters passed, downloads a page from the Internet and stores it in an output file. If it gets an error while retrieving the page it will output an error message. If run without any parameters it will report the correct method of running the program. The program is called `urldump.smp`, and there is a project already located in the `Projects\console` directory.

Before you run off and start devouring that program though, it would be a good idea to continue reading here. The a reason is that although that program will show you how its done, it won't be able to explain how it came to be in that form. That said, it is probably time to do just that.

First Steps

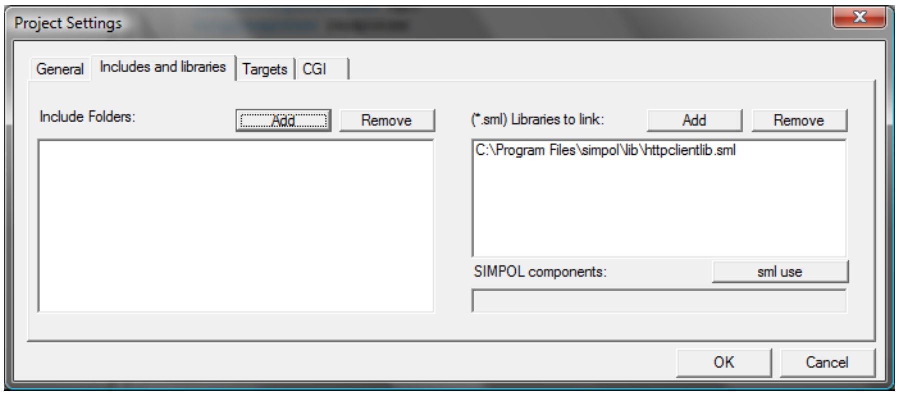
Since every SIMPOL program begins with the function `main()`, that is where we will start. The image below shows the beginning of the project. At this very early stage, there is not much there. The httpresponse type is also not in blue, but instead it is in black. That is a sign that the library is not yet part of the project.



```
function main(string sUrl, string sOutfile)
  string errtext
  integer e
  fsfileoutputstream fpo
  httpresponse response

  if sUrl <= ""
    errtext = usage()
  else
  end if
end function
```

Initial stage of the `urldump.smp` project.

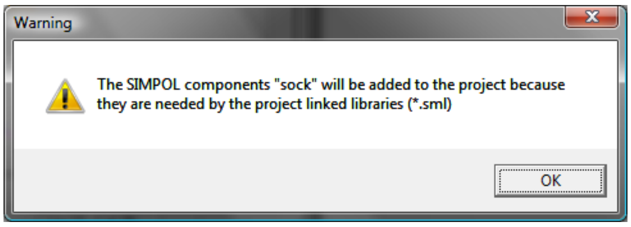
To resolve this, we can add the required library to the project. From the Project menu, select the Settings item to display the Project Settings dialog. Select the second tab, Includes and libraries, and then click on the Add button next to the (*.sml) Libraries to link; label. From there, enter the SIMPOL `lib` directory and pick the `httpclientlib.sml` file. The result should look like the image below:



C:\Program Files\simpol\lib\httpclientlib.sml

The Project Settings dialog after adding the `httpclientlib.sml` library.

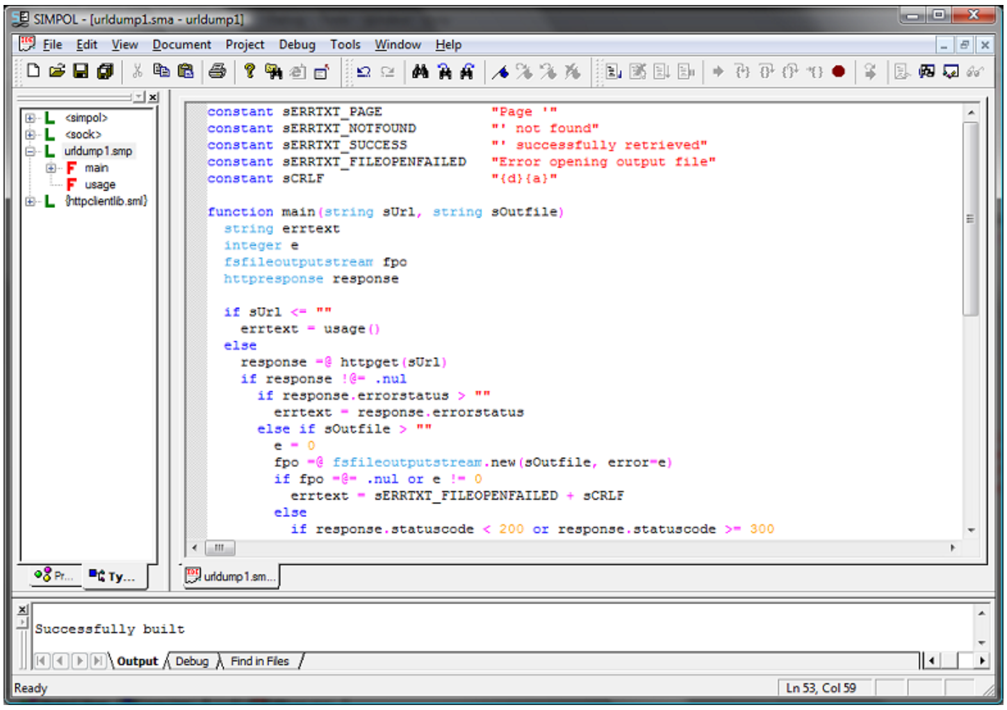
At this point, clicking on the OK button will result in a warning dialog being shown. This one warns us that the `httpclientlib.sml` library requires the SIMPOL component `sock` and therefore this will also be added to the project. This is quite handy, since otherwise the library wouldn't even work. The warning dialog looks like this:



The SIMPOL components "sock" will be added to the project because they are needed by the project linked libraries (*.sml)

The warning dialog shown when a library has been added that requires components that are not currently part of the project.

Depending on the size of the screen area on our computer, it may be useful to turn off a couple of windows while writing the program. This can be done from the View menu, by selecting the Call Stack and Variables items, for example. After a bit more code has been written, and with our new adjusted windows, the result might look like the following image.



```
constant sERRTXT_PAGE              "Page '"
constant sERRTXT_NOTFOUND          "' not found"
constant sERRTXT_SUCCESS           "' successfully retrieved"
constant sERRTXT_FILEOPENFAILED    "Error opening output file"
constant sCRLF                     "{d}{a}"

function main(string sUrl, string sOutfile)
  string errtext
  integer e
  fsfileoutputstream fpo
  httpresponse response

  if sUrl <= ""
    errtext = usage()
  else
    response =@ httpget(sUrl)
    if response !@= .nul
      if response.errorstatus > ""
        errtext = response.errorstatus
      else if sOutfile > ""
        e = 0
        fpo =@ fsfileoutputstream.new(sOutfile, error=e)
        if fpo =@= .nul or e != 0
          errtext = sERRTXT_FILEOPENFAILED + sCRLF
        else
          if response.statuscode < 200 or response.statuscode >= 300
```

```
Successfully built
```

The project in its more advanced state after also adjusting some of the windows for greater code visibility.

At this point, let's actually have a look at our first version of this program.

Example 4.1. Initial version of `urldump.sma`

```
constant sERRTXT_PAGE             "Page '"
constant sERRTXT_NOTFOUND         "' not found"
constant sERRTXT_SUCCESS          "' successfully retrieved"
constant sERRTXT_FILEOPENFAILED   "Error opening output file"
constant sCRLF                    "{d}{a}"

function main(string sUrl, string sOutfile)
  string errtext
  integer e
  fsfileoutputstream fpo
  httpresponse response

  if sUrl <= ""
    errtext = usage()
  else
    response =@ httpget(sUrl)
    if response !@= .nul
      if response.errorstatus > ""
        errtext = response.errorstatus
      else if sOutfile > ""
        e = 0
        fpo =@ fsfileoutputstream.new(sOutfile, error=e)
        if fpo =@= .nul or e != 0
          errtext = sERRTXT_FILEOPENFAILED + sCRLF
        else
          if response.statuscode < 200 or \
             response.statuscode >= 300
            errtext = sERRTXT_PAGE + sUrl + \
                      sERRTXT_NOTFOUND + sCRLF
          else
            errtext = sERRTXT_PAGE + sUrl + sERRTXT_SUCCESS + sCRLF
          end if
          fpo.putstring(.if(response.entitybody != .nul, \
                response.entitybody.getstring(1, .inf, 1), ""), 1)
        end if
      else
        if response.statuscode < 200 or response.statuscode >= 300
          errtext = sERRTXT_PAGE + sUrl + sERRTXT_NOTFOUND + sCRLF
        else
          errtext = ""
        end if
        errtext = errtext + .if(response.entitybody != .nul, \
                    response.entitybody.getstring(1, .inf, 1), "")
      end if
    end if
  end if
end function errtext


function usage()
```

```
  string s

  s = "smprun[32.exe] urldump.smp <url> <outputfile>{d}{a}"
end function s
```

Understanding the Code

Although there is not much to this program, it covers a number of concepts that are worth exploring. To begin with, the command line parameters are always string variables and they do not allow for default values, so to set those you will need to write some code for it. At the beginning of the program, there is a test for the `sUrl` variable. If it finds that no value has been passed, then it calls the `usage()` function. This approach makes it quite easy to both document how the program works and also inform the user when the parameters are not correct.

The next thing to note is the call to the `httpget()` function. That returns an httpresponse object (and should do so under all circumstances, so the following test for `.nul` may be unnecessary). The httpresponse object contains all the information that is a result of the attempt to retrieve the resource represented by the `sUrl` variable. Should there have been any unexpected problem with the retrieval of the resource then the errorstatus property would have some value greater than the empty string (`""`).

The remaining code simply checks whether the output is going to a file or if it will be output as part of the return value. In each case, it outputs the content of the entitybody property if the retrieval was successful (a value between 200 and 299 in the statuscode property) then a success string is returned, otherwise an error string.

Running Our Project

At this point we should build our project (**Ctrl**+**B** — Build). Now we can run the program, but if we want to try it in the IDE we will also need to define the argument that is being passed to the `main()` function. We can do that in the Project menu by selecting the Settings item again. In the first tab, in the Command line box, enter the URL:

> http://www.google.co.uk/search?hl=en&q=SIMPOL&btnG=Google
> +Search&meta=&aq=f&oq=

and then click OK to close the dialog.

Note

It is always a good idea to click on the Save All icon on the tool bar after making changes to the project's settings. This ensures that those changes are saved to the project file. If you don't, and the IDE crashes for some reason, you may lose the changes that you have made.

Now to run the program, press **Ctrl**+**E** (Execute). The result should be the page containing the Google search results for the key word "`SIMPOL`". The page will be messy and hard to read, since it normally returns as an unformatted stream of characters without any new line characters. To see a page that may look a little more familiar, try the URL "`www.simpol.com`". That should look like a fairly readable page of XHTML.

Improving Our Program

Although this program isn't bad, it might be useful if it were a little more flexible. One thing we might want to do is allow it to take parameters, so that it can do not only a `GET` operation, but also a `POST`. We could also decide to allow the program to output the header information that it received from the web server, which can be very handy when trying to debug routines that retrieve data from a web server. A method of handling and validating command line parameters might also be useful. Let's add support for not only the output file, but also a flag to decide if the header is output, and also support for passing variables through using the `POST` method.

As a first step, we can update our `usage()` function with the new information. The new version looks like this:

Example 4.2. Updated `usage()` Function

```
function usage()
  string s

  s = "smprun[32.exe] urldump.smp <url> [--outfile=<filename>] \
        [--showheader]{d}{a}"
  s = s + "    [--vars=<varlist>]{d}{a}{d}{a}"
  s = s + "    Where the vars need to already be URL-encoded and \
        if they violate the{d}{a}"
  s = s + "    shell rules they will also need to be escaped to \
        be hidden from the{d}{a}"
  s = s + "    shell. It is recommended to place quotes around \
        the --vars= entry.{d}{a}"
  s = s + "    The equals sign and what follows CANNOT allow \
        spaces! If necessary,{d}{a}"
  s = s + "    surround any entry with quotes.{d}{a}"
end function s
```

Now that we have decided what the parameters are going to be (and incidentally also the format), we can add the code to handle the parameters. This is probably best done using a specifically designed data type. This will allow us to offload most of the work to the type itself, without cluttering our existing `main()` function with all the associated code. It will also make it easier to lift it and use it again in another program, or even in the future to create a more versatile type that is more universal. Let's see what that code looks like:

Example 4.3. The parameters Type

```
type parameters
  embed
  string outfilename
  boolean showheader
  string variables
  string operation

  reference
  function getparam
end type


function parameters.new(parameters me)
  me.outfilename = ""
  me.showheader = .false
  me.variables = ""
  me.operation = "GET"
end function me
```

```
function parameters.getparam(parameters me, string parameter)
  if parameter <= ""
    // do nothing
  else if .like1(parameter, "--outfile=*")
    me.outfilename = .substr(parameter, .instr(parameter, "=") + \
                             1, .inf)
  else if .like1(parameter, "--showheader")
    me.showheader = .true
  else if .like1(parameter, "--vars=*")
    me.variables = .substr(parameter, .instr(parameter, "=") + \
                           1, .inf)
  end if
end function
```

The parameters type has properties to store all the information that we will use for this call to the program. It defaults to running in `GET` mode, and will not return the header from the web server. The way the `getparam()` method has been coded requires that each parameter that has a value component must be separated from the value by an equals (=) sign and no white space to either side. Part of the reason for this is that allowing white space would require a more complex algorithm, since each of the white space separated items would arrive as separate parameter values from the shell to the `main()` function.

We now have a method of handling the various parameters to the program, and one of the nice features of this approach is that the order of the parameters does not matter. The only parameter that has a fixed position is the URL itself, since it must be first. Using this approach requires some changes to the `main()` function as well. Let's have a look at those now.

Example 4.4. The Final Version of the `main()` Function

```
function main(string sUrl, string param1, string param2, \
               string param3)
  string errtext
  integer e
  fsfileoutputstream fpo
  httpresponse response
  parameters params

  e = 0
  if sUrl <= ""
    errtext = usage()
  else
    params =@ parameters.new()
    params.getparam(param1)
    params.getparam(param2)
    params.getparam(param3)

    errtext = ""
    if params.variables > ""
      response =@ httppost(sUrl, params.variables)
    else
      response =@ httpget(sUrl)
    end if

    if response !@= .nul
      if response.errorstatus > ""
```

```
        errtext = response.errorstatus
      else if params.outfilename > ""
        fpo =@ fsfileoutputstream.new(params.outfilename, error=e)
        if fpo =@= .nul or e != 0
          errtext = sERRTXT_FILEOPENFAILED + sCRLF
        else
          if response.statuscode < 200 or \
             response.statuscode >= 300
            errtext = sERRTXT_PAGE + sUrl + sERRTXT_NOTFOUND + \
            sCRLF
          else
            errtext = sERRTXT_PAGE + sUrl + sERRTXT_SUCCESS + sCRLF
          end if
          fpo.putstring(.if(params.showheader, \
                            makenotnull(response.fullheader) + \
                            sCRLF + sCRLF, "") + \
                        .if(response.entitybody != .nul, \
                            response.entitybody.getstring(1, \
                            .inf, 1),""), 1)
        end if
      else
        if response.statuscode < 200 or response.statuscode >= 300
          errtext = sERRTXT_PAGE + sUrl + sERRTXT_NOTFOUND + sCRLF
        else
          errtext = ""
        end if
        errtext = errtext + makenotnull(.if(params.showheader, \
                                            response.fullheader + \
                                            sCRLF + sCRLF, "")) + \
                        .if(response.entitybody != .nul, \
                            response.entitybody.getstring(1, \
                            .inf, 1),"")
      end if
    end if
  end if
end function errtext
```

As we can see from the previous code, not a lot has changed from the original version. We now support the `POST` operation if we were given variables (which must be URL-encoded when they are passed in). We can also optionally return the entire header from the web server if requested to do so. All of the actual handling of the parameters is done by the parameter type and its `getparam()` method.

Running the Final Version

Now that all our coding is done (the final coded version of this example can be found in the supplied program samples as a console project called `urldump`. This is the command line we will use to try out the new features:

Example 4.5. Sample Command

```
urldump.smp "wwwx.cs.unc.edu/~jbs/aw-wwwp/docs/resources/perl/
  perl-cgi/programs/cgi_stdin.cgi" --showheader "--vars=name=Joe&
  textarea=Cool&radiobutton=middleun&checkedbox=pizza&
  selectitem=hamburgers"
```

Note

The URL in the previous command was found while searching on the Internet. It may or may not be there forever, but it is greatly appreciated for providing an opportunity to test the `POST` operation in this program. Eventually we may produce a sample program running from our own web site but anticipate the likely web load of a few people trying this out will not greatly inconvenience the university site.

The result of running this new version of the program with the command line parameters shown above, can be seen in the section below:

```
-----------------  20:41:34 13/08/2009  -----------------
Executing "X:\simpol\projects\console\urldump\bin\urldump.smp" ...
--------------------- program result --------------------
HTTP/1.1 200 OK
Date: Thu, 13 Aug 2009 19:41:35 GMT
Server: Apache/2.2.3 (Red Hat)
Connection: close
Content-Type: text/html

<HEAD>
<TITLE>stdin vars.</TITLE>
<H1>Print CGI STDIN Variables</H1>
</HEAD>
<BODY>
<HR>
<H3>STDIN Variables</H3>
<UL>
<LI>radiobutton = middleun
<LI>checkedbox = pizza
<LI>name = Joe Bloggs
<LI>selectitem = hamburgers
<LI>textarea = Cool form dude
</UL>
</BODY>

---------------------------------------------------------
Successfully executed
```

Summary

In this chapter we have developed a command line program to retrieve a page across the Internet using both `GET` and `POST`. We extended the initial version of the program to also take named parameters in any order. The techniques learned here could also be applied in other programs. The parameter handling can be reused in other command line programs. The use of the `httpclientlib.sml` library could be added to a web server or desktop program to retrieve information from another location on the Internet, such as currency exchange rates, stock market values, etc.

Chapter 5. Dialog-Style Programs

What's a Dialog Program?

A dialog program in the sense used in this book refers to typically smaller, less complicated programs that typically only require a single window, without a menu or tool bar. These types of programs are common of smaller utilities. They are not typically connected to a database table, though that doesn't mean they can't be. Also, the same approach used to create a dialog-style program can be used to create dialog-style functionality as part of a larger program. In some cases, the stand alone utility program can easily be converted to provide its functionality within the context of a larger program.

The example program that we will use in this chapter was selected to fulfill a number of goals:

- The sample should provide some useful functionality
- It should demonstrate the use of the SIMPOL Form Designer
- It should use generated form source code
- It should be possible to incorporate the result in another program if desired

The Sample Program

The sample program we will use in this section is called SIMPOL Color Lab. While thinking about the program, it was decided to make something that would let the user see the color that corresponds to a specific web color value. These are normally specified in the format: `#A0B0C0` where the first character is an indicator signifying that the following number is in the hexadecimal format and the following values are interpreted as the three RGB values between 0 and 255 (or in this case between 0 and FF). To make the tool more interesting, the user should also be able to enter a decimal value, or the values for the three color components: red, green, and blue. These can be entered by hand or by moving a sliding widget.

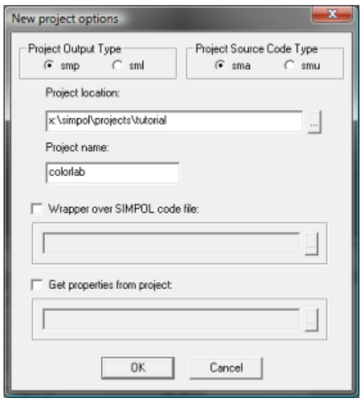
There are many different design approaches that can be taken with development of a software project, some of them more formal than others. This sort of project lends itself well to a fairly informal and interactive approach. To get this project moving, it would be a good idea to start with the layout and look of the dialog that the user will see. Once the basic design is done, it will only require the additional work to show the dialog and react to what happens when it is used.

Note

The source code for this project is included in the Superbase NG product and can be found in the directory `\Projects\tutorial\colorlab`.

Creating the Project

As a first step, it would be a good idea to create the project. That will provide us with a location for storing the form once we finish designing it. We won't actually write any code until later, though. Start the SIMPOL IDE and once it is running from the menu, select File → New Project. In the dialog window that is then displayed, select an appropriate location for the project, and give it the name `colorlab`. See the New Project window image shown here:


New project options

Project Output Type: [✓ smp] sml

Project Source Code Type: [✓ sma] smu

Project location: x:\simpol\projects\tutorial

Project name: colorlab

☐ Wrapper over SIMPOL code file:

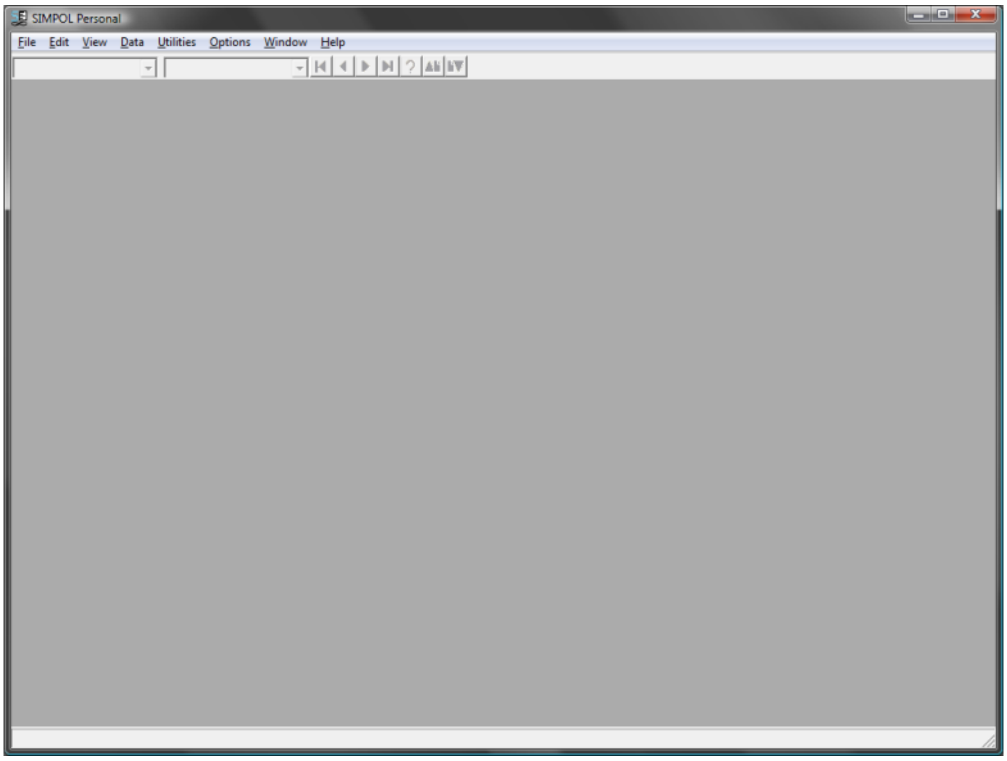
☐ Get properties from project:

Image of the SIMPOL IDE New Project dialog.

Click on the OK button to create the project. Remember this location, we will use it save our form design later. For now, minimize or close the IDE; we won't need it anymore for a while yet.

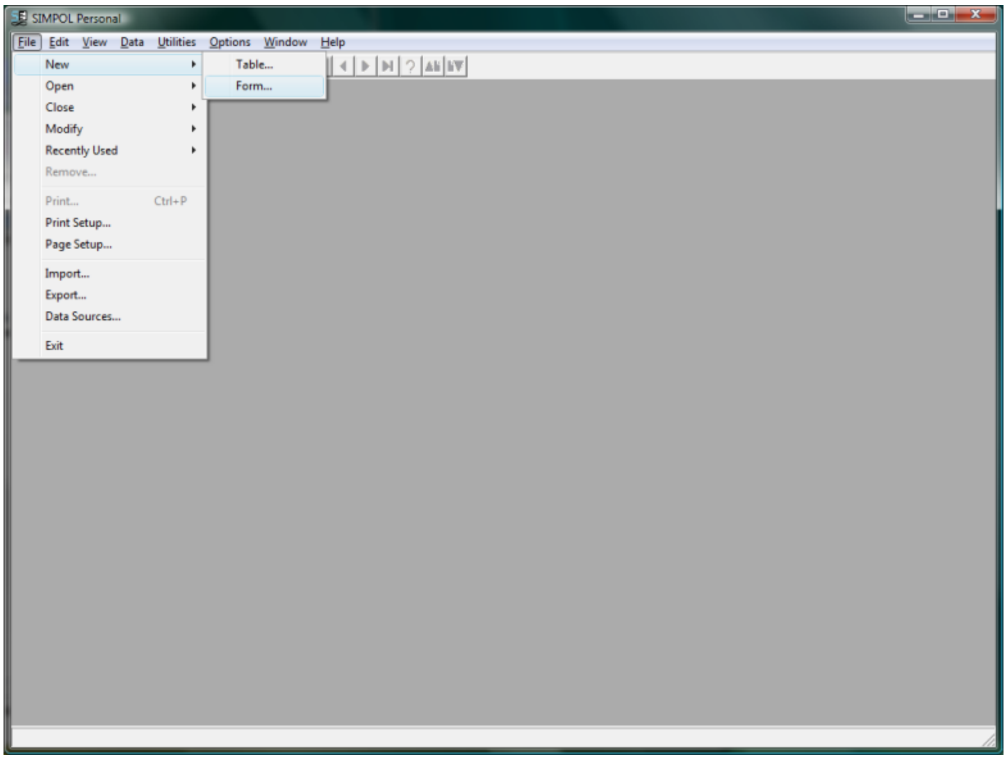
Creating the Design

To create the design, start Superbase NG Personal (it can be found in the main program group for Superbase NG). Once it has started running, it should look something like this:



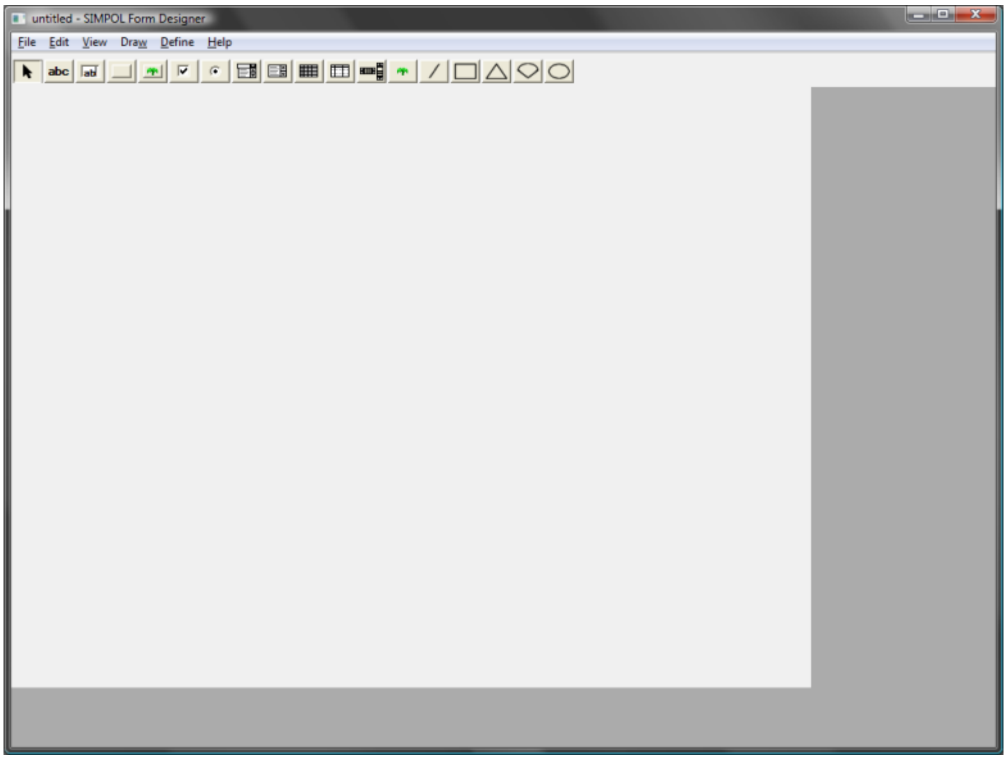
Superbase NG Personal just after being started.

Then select File → New → Form from the Superbase NG Personal program to open the Form Designer and create a new form.



Starting the Form Designer from the Superbase NG Personal program.

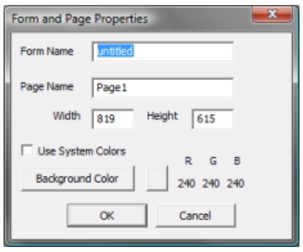
Once that has been done, the SIMPOL Form Designer will open and present a picture similar to the one that follows. It contains a blank form sized to a percentage of the size of the screen, with no controls on it.


untitled - SIMPOL Form Designer
File Edit View Draw Define Help

The SIMPOL Form Designer with a new blank form.

Setting the Stage

Now that we have a blank form, it is a good idea to set a few default properties and to give the form a preliminary size. Select File → Page Setup… from the menu, or double-click the left mouse button on the form to display the Form and Page Properties dialog, which will look something like the one below:


Form and Page Properties

Form Name: untitled

Page Name: Page1

Width: 819 Height: 615

☐ Use System Colors

Background Color R 240 G 240 B 240

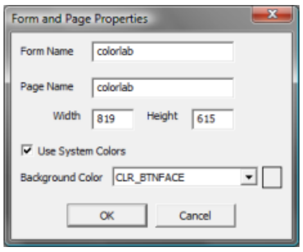
The Form and Page Properties dialog.

Change the both the Form Name and the Page Name to `colorlab`. Now click on the Use System Colors check box, which will default to the CLR_BTNFACE entry. What this does is allow the form to inherit the system settings for the color scheme. By selecting this setting, the settings for the controls will also default to using system colors. For now, leave the page size as it is.

Note

It is not necessary to work using this approach. If you want to specifically set the colors used for various parts of the form, feel free to do so. Just realize that users expect their applications to look like the other applications that they use and if they do not, they may react negatively to a program, or consider it to be *unprofessional*.

After all of our changes, the resulting dialog will look roughly like this:


Form and Page Properties

Form Name: colorlab

Page Name: colorlab

Width: 819 Height: 615

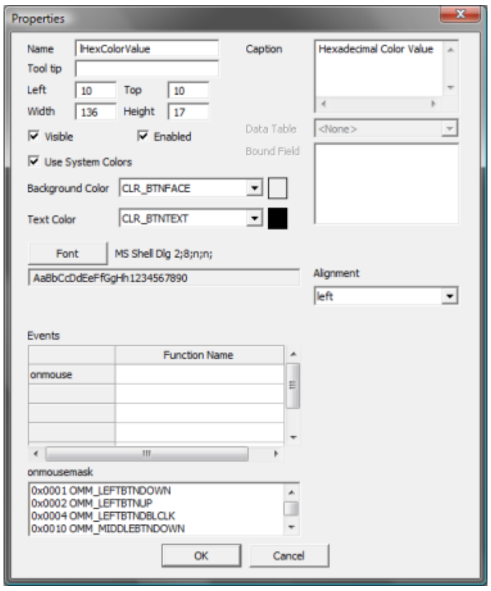
☑ Use System Colors

Background Color: CLR_BTNFACE

The Form and Page Properties dialog after changes.

Adding the Controls to the Form

It is now time to start adding controls to the form. Start by selecting the abc button from the tool bar, or by selecting Draw → Text from the menu, and then using the mouse click the left mouse button and while holding the button down, drag a rectangular outline for the form label. Release the mouse button when the rectangle represents the area desired for the label. When the button is released, the Properties dialog window is shown. For now, just set the Name field to "`lHexColorValue`" and the Caption to "`Hexadecimal Color Value`", leaving the rest unchanged. The content will be similar to what is shown here:



Name	lHexColorValue
Tool tip	
Left	10
Top	10
Width	136
Height	17
Visible	✓
Enabled	✓
Use System Colors	✓
Background Color	CLR_BTNFACE
Text Color	CLR_BTNTEXT
Caption	Hexadecimal Color Value
Data Table	<None>
Bound Field	
Font	MS Shell Dlg 2;8;n;n;
	AaBbCcDdEeFfGgHh1234567890
Alignment	left

Events

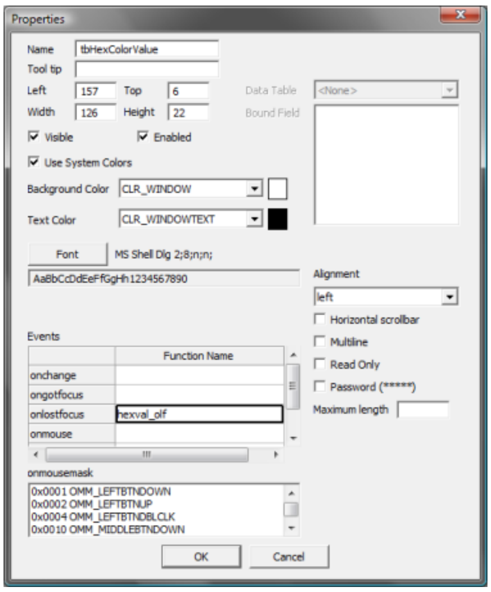
	Function Name
onmouse	

onmousemask

```
0x0001 OMM_LEFTBTNDOWN
0x0002 OMM_LEFTBTNUP
0x0004 OMM_LEFTBTNDBLCLK
0x0010 OMM_MIDDLEBTNDOWN
```

The Properties dialog for a label.

Now add an editable text box to the form, by selecting the ab| button from the tool bar, or by selecting Draw → Editable Text from the menu. In the resulting dialog box, set the Name to `tbHexColorValue` and in the Events grid, set the value of the onlostfocus Function Name entry to `hexval_olf`. This sets the name of the function that is to be called when the event occurs. The dialog should then look similar to the one below:



Name	tbHexColorValue
Tool tip	
Left	157
Top	6
Width	126
Height	22
Visible	✓
Enabled	✓
Use System Colors	✓
Background Color	CLR_WINDOW
Text Color	CLR_WINDOWTEXT
Data Table	<None>
Bound Field	
Font	MS Shell Dlg 2;8;n;n;
	AaBbCcDdEeFfGgHh1234567890
Alignment	left
Horizontal scrollbar	☐
Multiline	☐
Read Only	☐
Password (*****)	☐
Maximum length	

Events

	Function Name
onchange	
ongotfocus	
onlostfocus	hexval_olf
onmouse	

onmousemask

```
0x0001 OMM_LEFTBTNDOWN
0x0002 OMM_LEFTBTNUP
0x0004 OMM_LEFTBTNDBLCLK
0x0010 OMM_MIDDLEBTNDOWN
```

The Properties dialog for an editable text box.

Note

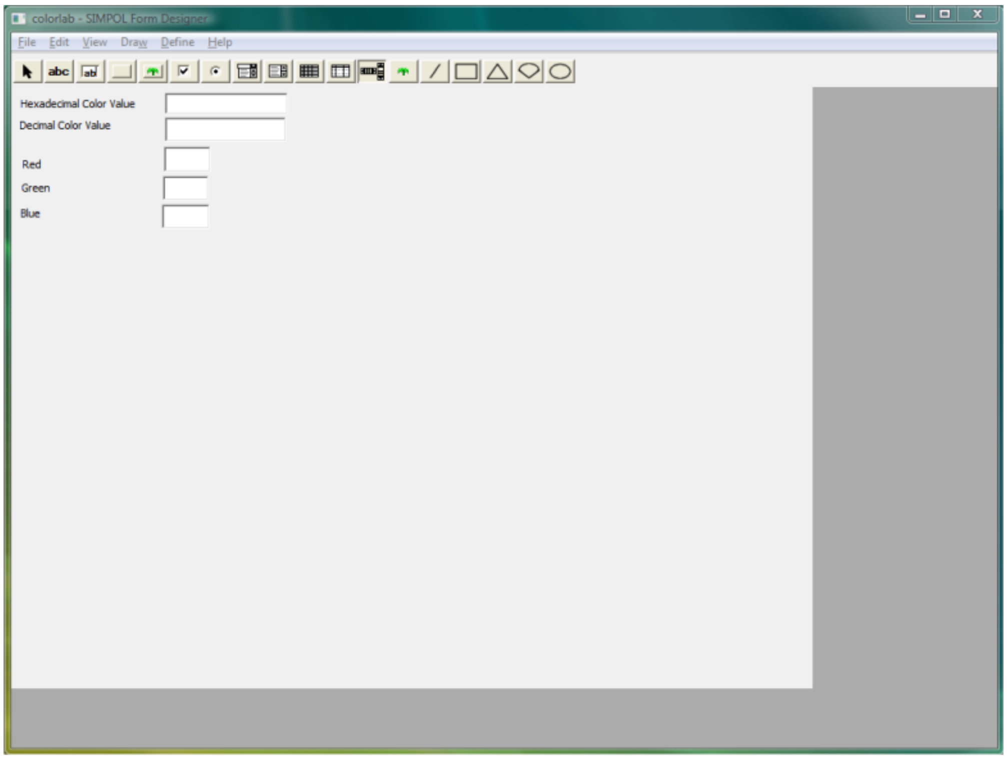
In the preceding text the two elements have been assigned the names: `lHexColorValue` and `tbHexColorValue`. It isn't absolutely essential to name the elements of the dialog, the Form Designer will do it for you, but the names won't be very meaningful. Whenever you expect to actually need to change the content or read the content (or the state – visible, enabled, etc.) or a control, it is a good idea to give it a meaningful name. Also, using a convention for the names will help you remember what type of control you are dealing with in the code. A common convention used in the examples is:

Table 5.1. Control Naming Conventions

Prefix	Explanation
l	Label – used for wxformtext objects
tb	Text Box – used for wxformedittext objects
b	Button – used for wxformbutton objects
bb	Bitmap Button – used for wxformbitmapbutton objects
ck	Check Box – used for wxformcheckbox objects
ob	Option Button – used for wxformoption objects
lb	List Box – used for wxformlist objects
cb	Combo Box – used for wxformcombo objects
g	Grid – used for wxformgrid objects
sb	Scroll Bar – used for wxformscrollbar objects
b	Bitmap – used for wxformbitmap objects
l	Line – used for wxgraphicline objects
r	Rectangle – used for wxgraphicrectangle objects
t	Triangle – used for wxgraphictriangle objects
a	Arc – used for wxgraphicarc objects
e	Ellipse – used for wxgraphicellipse objects

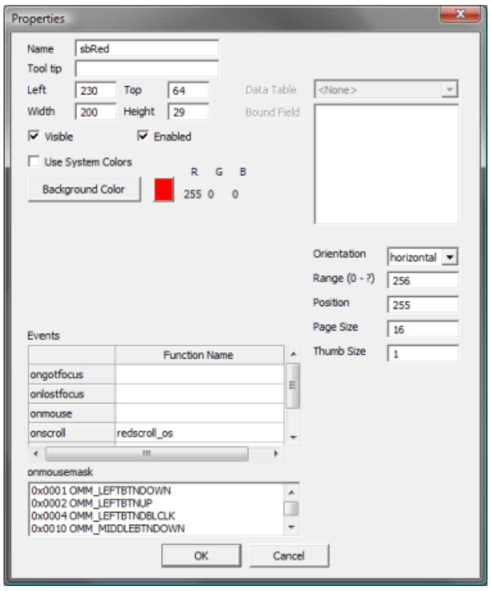
You may notice that some of the values are used more than once, such as `b` for both bitmaps and buttons. Although it may seem like it could be confusing, in practice the name that goes with the type identifier tends to make a clear distinction. It is also a good practice to use the same base name for a label and an edit control that are meant to go together, like in the example above.

Now we add some more labels and edit controls for: the decimal color value, and the red, green, and blue color values. The resulting form looks like this so far:



The state of the form after placing two controls.

The form isn't particularly pretty yet, but before we do any cleanup, it may be useful to just plant the rest of the controls on the form. To make things visually more interesting we will use three horizontal scroll bars for controlling the three color components. Select the scrollbars control from the controls palette on the tool bar, or select the Draw → Scrollbar item from the menu. Drag a horizontal rectangle on the form and then set the properties in the dialog as shown in the image below:



Name	sbRed
Tool tip	
Left	230
Top	64
Width	200
Height	29
Data Table	<None>
Visible	✓
Enabled	✓
Use System Colors	☐
R G B	255 0 0
Orientation	horizontal
Range (0 - ?)	256
Position	255
Page Size	16
Thumb Size	1

Events

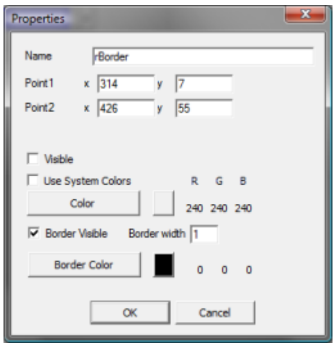
	Function Name
ongotfocus	
onlostfocus	
onmouse	
onscroll	redscroll_os

onmousemask

```
0x0001 OMM_LEFTBTNDOWN
0x0002 OMM_LEFTBTNUP
0x0004 OMM_LEFTBTNDBLCLK
0x0010 OMM_MIDDLEBTNDOWN
```

The Properties dialog for a scrollbar.

Please note that the background color of the red scrollbar was set to red. Now do the same again for the green and the blue scrollbars. Once they are on the form, add a rectangle to the form (to show the actual color value). To do this, select the appropriate control from the tool bar or the menu, then drag a rectangle onto the form. The Properties dialog for graphics is considerable simpler in design than that for form controls. Set the values as shown in the image that follows:



Properties

Name: rBorder

Point1 x 314 y 7

Point2 x 426 y 55

☐ Visible

☐ Use System Colors

Color — R 240 G 240 B 240

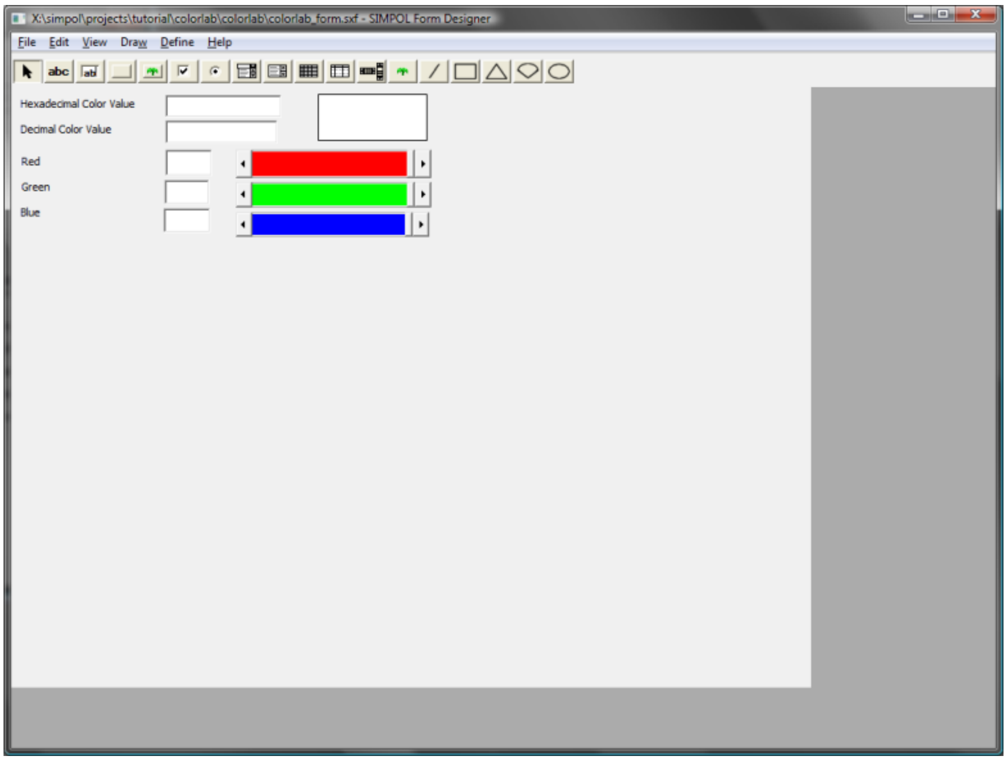
☑ Border Visible — Border width 1

Border Color — 0 0 0

OK Cancel

The Properties dialog for a rectangle.

Now that we have added much of the form content, let's take a look at the current state of our form.



X:\simpol\projects\tutorial\colorlab\colorlab\colorlab_form.sxf - SIMPOL Form Designer

Hexadecimal Color Value

Decimal Color Value

Red

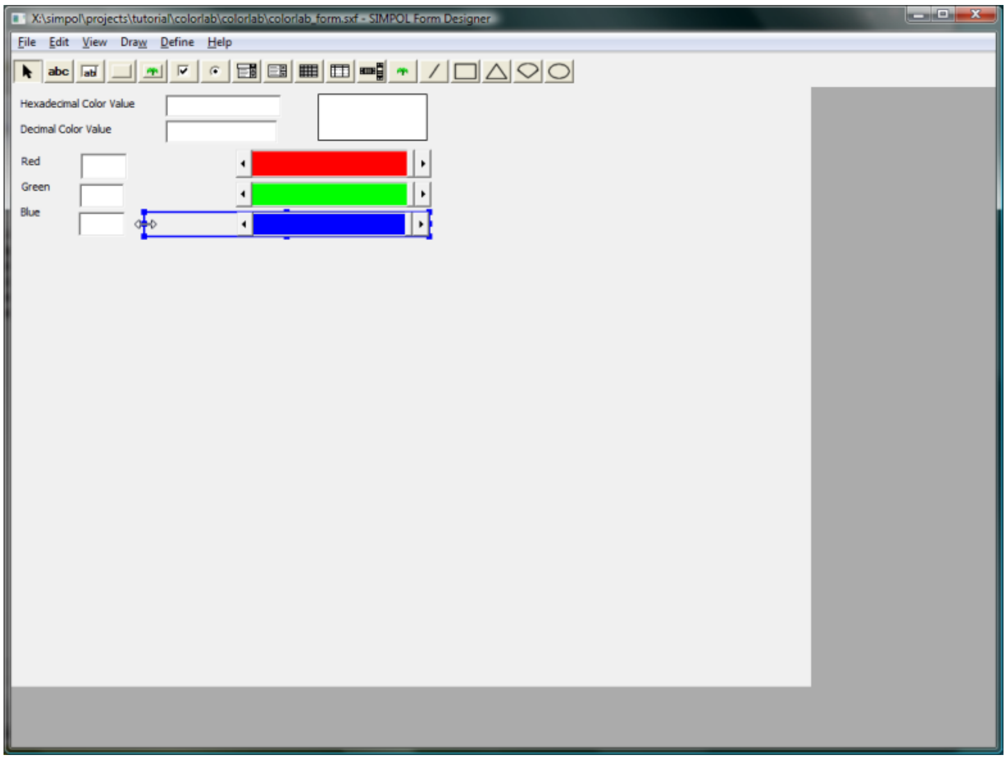
Green

Blue

The state of the form after adding most of the controls.

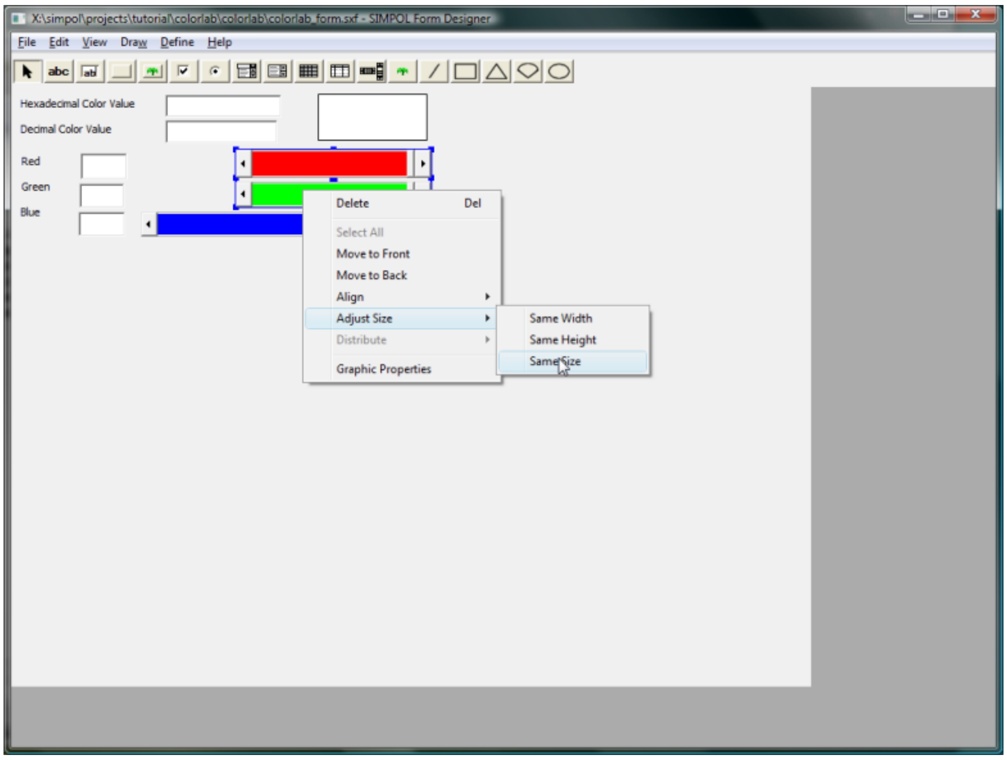
Cleaning Things Up

The form isn't looking too bad, but it just doesn't work very cleanly from a design perspective, so we are going to rearrange things a bit. Make sure the Arrow button is selected, and then drag a rectangle around the labels for the red, green, and blue boxes. Make sure they are all sized to not take up too much space beyond what they require (use the sizing handles for this). Now drag select the edit controls and move them over to the left. Resize the blue scrollbar to take up more of the space to the left that has now been made available. The result will look something like the following:


X:\simpol\projects\tutorial\colorlab\colorlab\colorlab_form.sxf - SIMPOL Form Designer
File Edit View Draw Define Help
Hexadecimal Color Value
Decimal Color Value
Red
Green
Blue

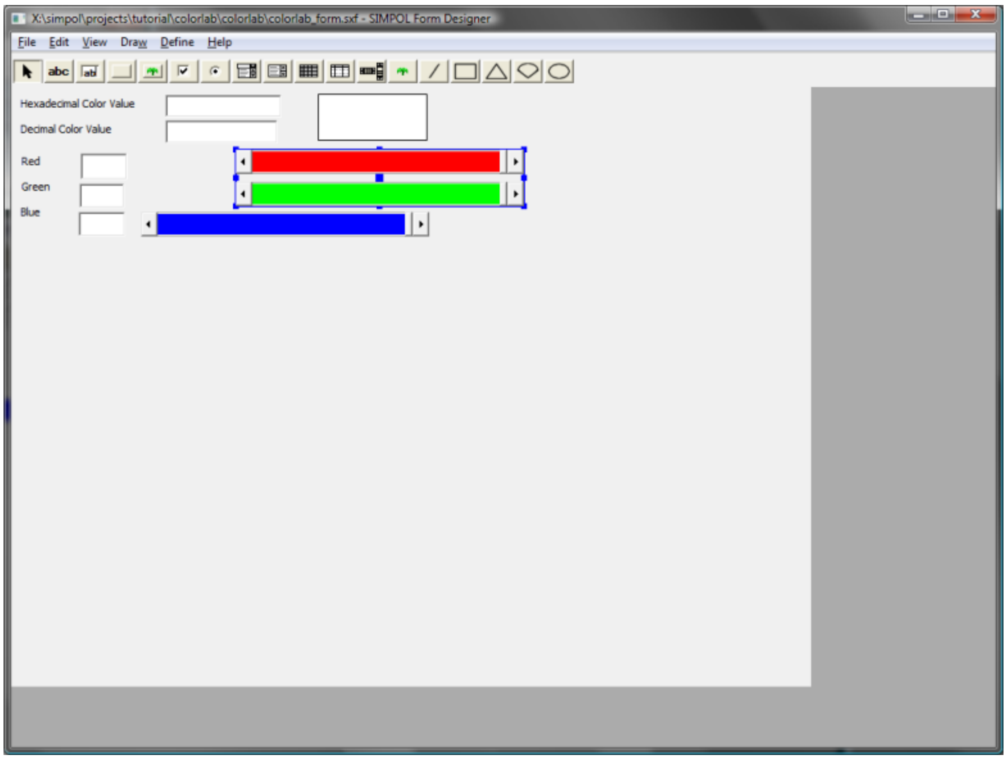
The state of the form after moving the color edit controls and during resizing of the blue scrollbar.

Now drag select the other two scrollbars and after clicking on the selected area with the right mouse button, select Adjust Size → Same Size, then click on the blue scrollbar that was just resized.



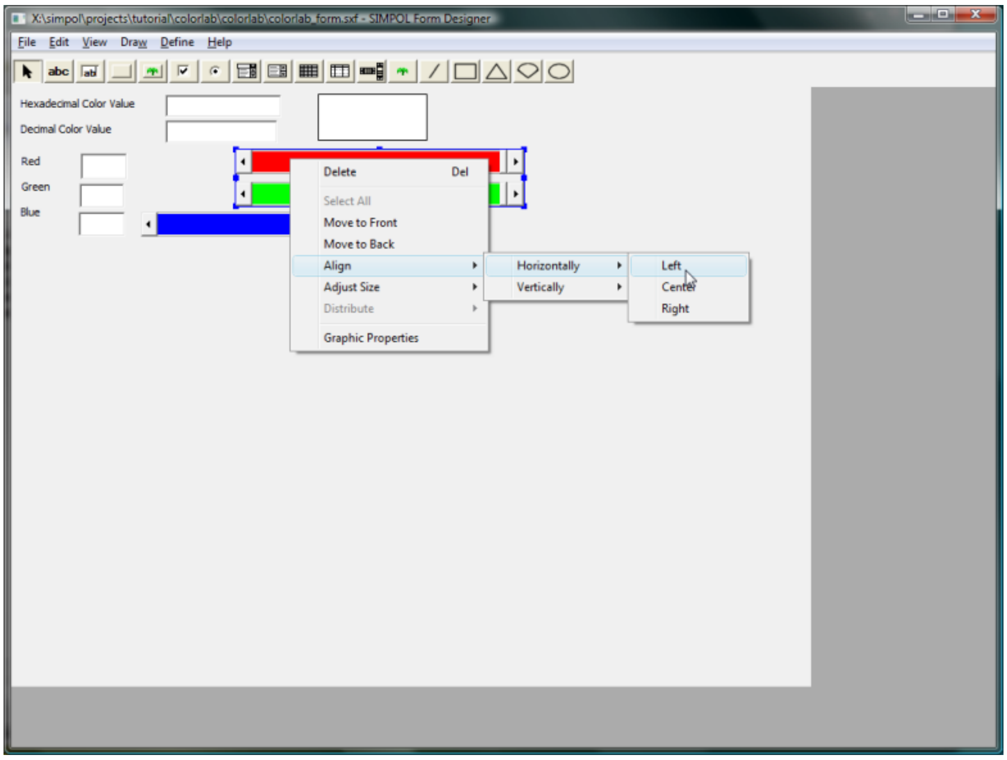
Showing the right-mouse pop-up menu to resize the selection.

This will then resize the selected scrollbars to the same size as the blue one as can be seen in the following image.

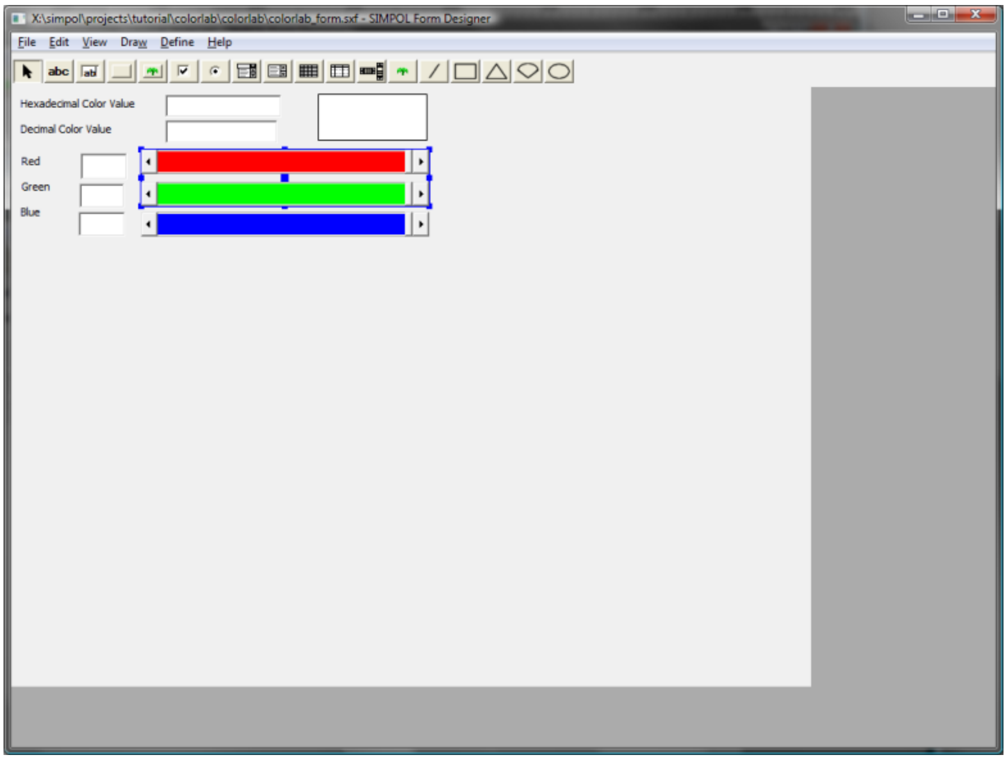

X:\simpol\projects\tutorial\colorlab\colorlab\colorlab_form.sxf - SIMPOL Form Designer
File Edit View Draw Define Help
Hexadecimal Color Value
Decimal Color Value
Red
Green
Blue

The resulting form after the scrollbars have been resized to match.

We have corrected the size, but they are still in the wrong place. By again clicking with the right mouse button on the still selected items, select the Align → Horizontally → Left choice and then click on the blue scrollbar. That will realign the scrollbars so that they are all at the same horizontal position on the form. The image that follows shows the menu selection, and the one after that the result.

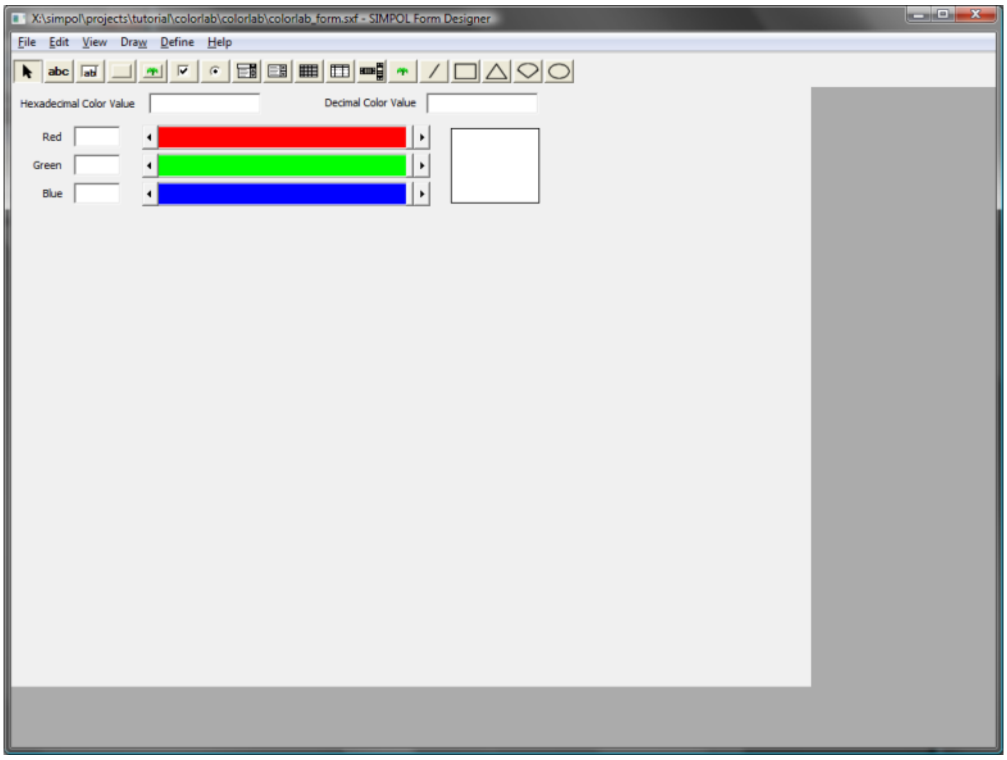


Showing the right-mouse pop-up menu for aligning the selection.


X:\simpol\projects\tutorial\colorlab\colorlab\colorlab_form.sxf - SIMPOL Form Designer
File
Edit
View
Draw
Define
Help
Hexadecimal Color Value
Decimal Color Value
Red
Green
Blue

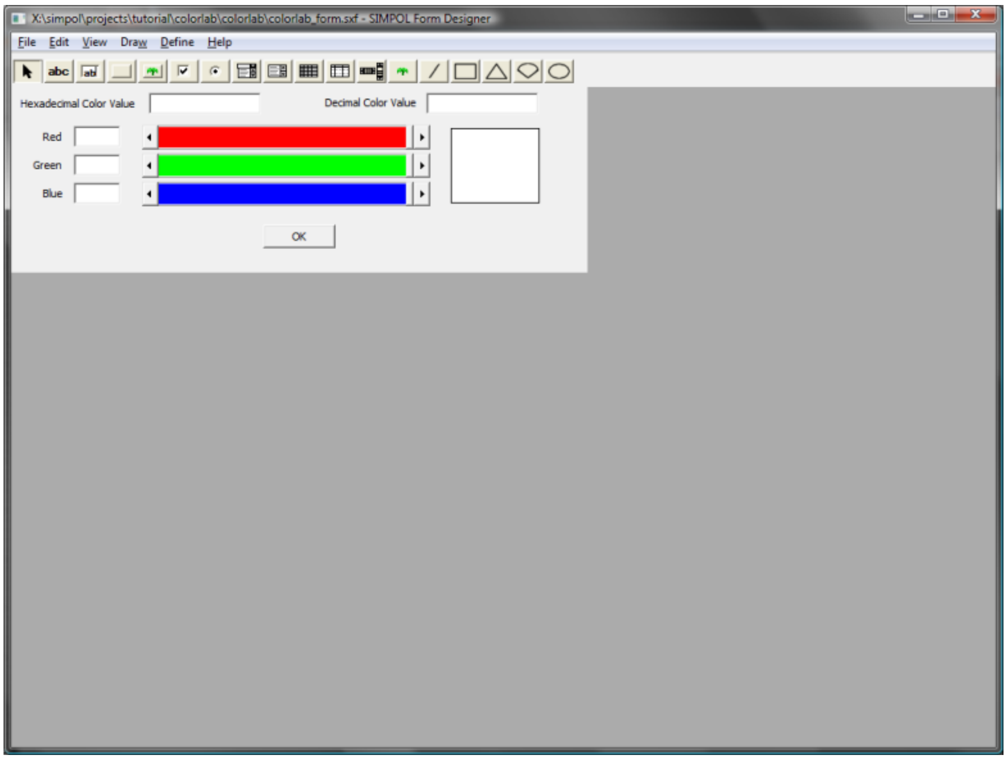
The resulting form after the scrollbars have been realigned.

Now, using the resizing, distribution, and alignment tools, plus the ability to select and then drag the selection around, the form is going to be rearranged. The final result after playing around to get a look that works, can be seen below:


X:\simpol\projects\tutorial\colorlab\colorlab\colorlab_form.sxf - SIMPOL Form Designer
File
Edit
View
Draw
Define
Help
Hexadecimal Color Value
Decimal Color Value
Red
Green
Blue

The resulting form after it has been rearranged.

The form is nearly finished. What would be useful is to provide the user with an OK button as an alternate way of closing the program. Also we should resize the form to fit around the contents without a lot of extra space. The final resulting form looks like this:


X:\simpol\projects\tutorial\colorlab\colorlab\colorlab_form.sxf - SIMPOL Form Designer
File
Edit
View
Draw
Define
Help
Hexadecimal Color Value
Decimal Color Value
Red
Green
Blue
OK

The final version of the form.

Saving the Form

To save the form in a format that can be reopened and modified, select File → Save As → Form… Save the form as `colorlab_form.sxf` in the project directory that we created earlier in the chapter. The best location would be the `colorlab\colorlab` directory, since that is also where our source directory is located.

For the purpose of this project, we are going to use the form as source code, so let's also save it off as SIMPOL source code. To do that, select File → Save As → wxform Program from the menu. Save the program as `colorlab_form.sma`. This should also be saved to the source directory fro our project. We will be using it in the next section.

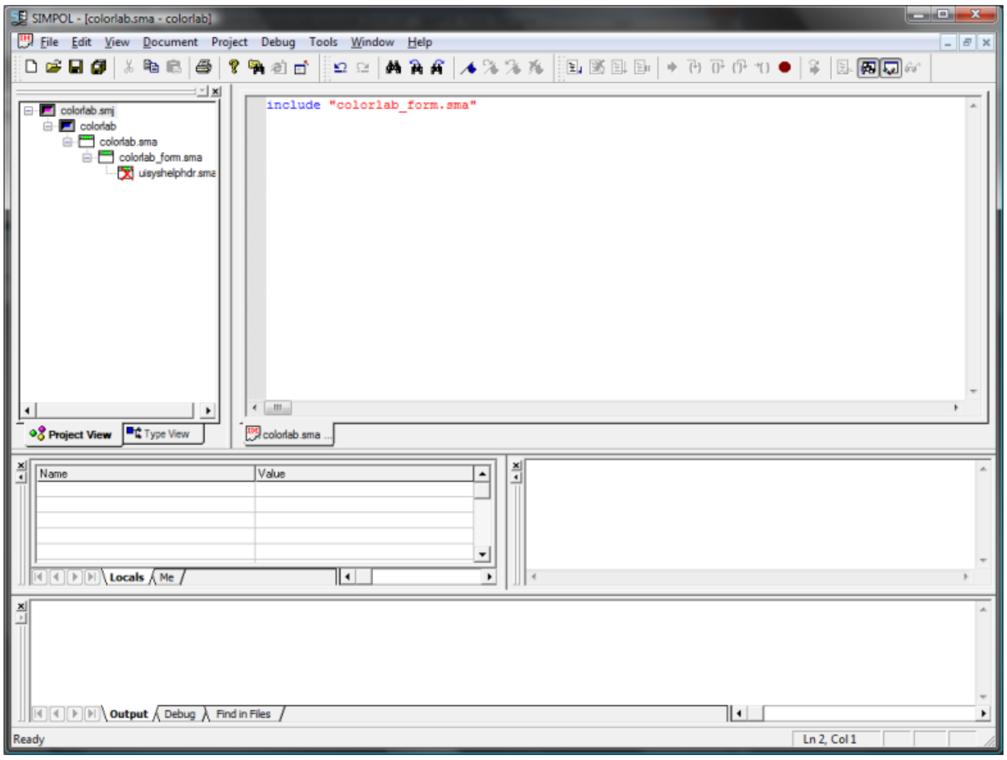
That's it! The design portion of our project is done. Now it is time to get on with turning our form into a working program.

Adding the Form Source to the Project

Now that the design portion is done, we can start actually getting the program running. If it is still open, Superbase NG Personal can be closed. Now switch to our minimized IDE from earlier in the chapter (or reopen the project if it was closed). The first thing we'll do is include the form source code:

```
include "colorlab_form.sma"
```

After adding this line, save the file (**Ctrl**+**S** or select File → Save from the menu). You will notice that a dependent entry for the file `colorlab_form.sma` appears in the Project View tree on the left. To see what that looks like, examine the image that follows.

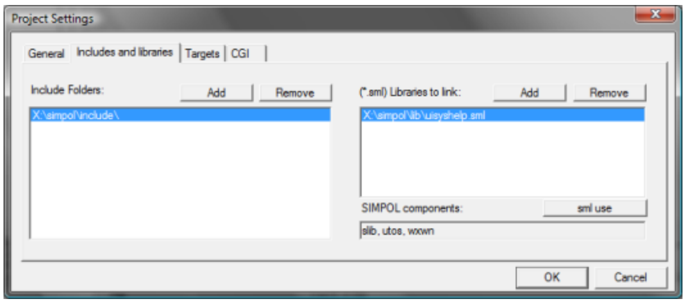


```
include "colorlab_form.sma"
```

Ready　Ln 2, Col 1

The SIMPOL IDE after adding the include entry to the project.

Also, another dependent entry appears below that, with a red X over the icon. The source file `uisyshelphdr.sma` is included by the form source code, but is not visible to the project. To resolve that, we need to add the `include` directory to the project. We also need to add a library that is used together with that include file. The library is called `uisyshelp.sml`. To do this, from the menu select Project → Settings. Then select the second tab, Includes and Libraries. On the left side click on Add and then locate and select the `include` directory from the SIMPOL installation. On the right side, also click the Add button. Select the `uisyshelp.sml` file to add it to the list of libraries. The result can be seen in the following image.



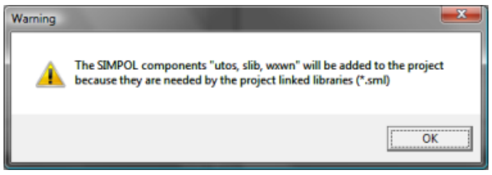
Include Folders: X:\simpol\include\

(*.sml) Libraries to link: X:\simpol\lib\uisyshelp.sml

SIMPOL components: slib, utos, wxwn

The Project Settings dialog after adding items.

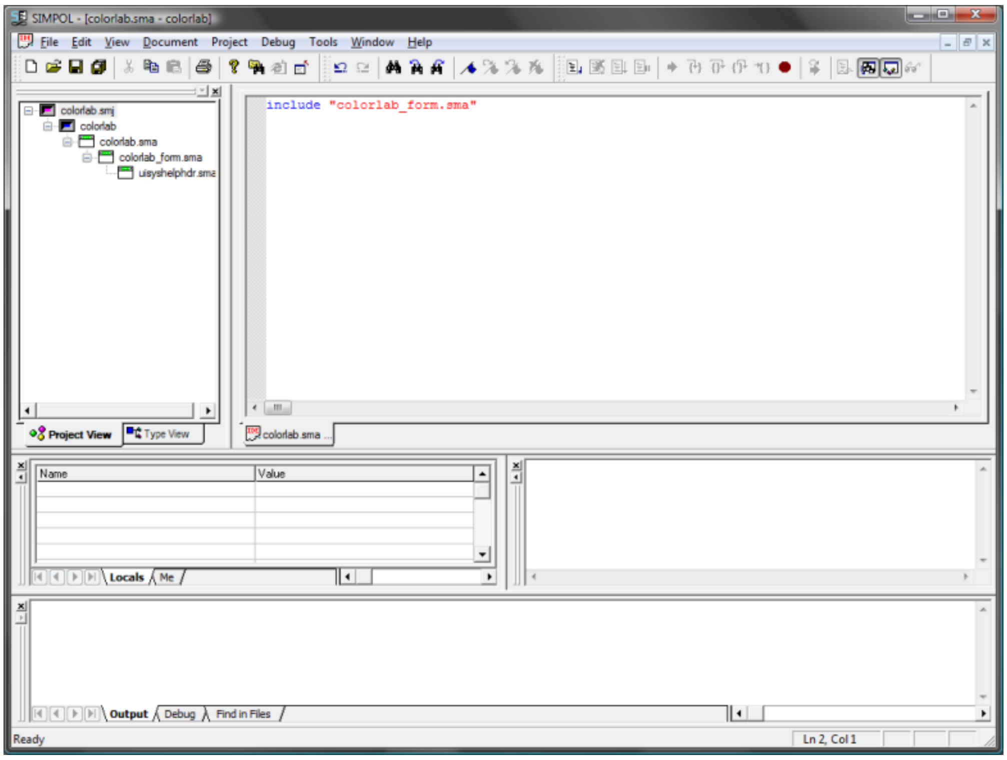
Now click on the OK button and that will show a warning dialog informing you that three additional components will be added to the project: `utos, slib, wxwn`. See the equivalent image hereafter.



The SIMPOL components "utos, slib, wxwn" will be added to the project because they are needed by the project linked libraries (*.sml)

The Project Settings warning dialog adding components.

This is done by detecting the required components from the library. Without those components the library would not work correctly. As can be seen from the following screen shot, once the update to the Include Folders has been added, the IDE now shows the entry in the Project View with a normal icon; the red X is gone.



```
include "colorlab_form.sma"
```

The SIMPOL IDE showing the updated project.

Setting Up the Program

The first step with getting the program running is to write the code that initializes the program, creates the form, creates the dialog window, and then waits for events. As it turns out, this does not require an excessive amount of program code.

Example 5.1. The `main()` Function of the colorlab Program

```
include "colorlab_form.sma"

constant iSTARTCOLOR  0xffffff

function main()
  wxform f
  wxdialog d
  integer e

  e = 0
  f =@ colorlab(error=e)
  if f =@= .nul
    errormsg(e)
  else
    d =@ wxdialog.new(1, 1, innerwidth=f.width, \
                      innerheight=f.height, \
                      visible=.false, \
                      captiontext="SIMPOL Color Lab", \
                      error=e)
    if d =@= .nul
      errormsg(e)
    else
      adjustformcolorvals(f, iSTARTCOLOR)
      f.setcontainer(d)
      centerdialogonparent(d)
      d.processmodal(.inf)
    end if
  end if
end function
```

The code above is fairly straightforward. It calls the generated `colorlab()` function to create the form. If anything goes wrong, it calls the `errormsg()` function if anything goes wrong. Assuming that the form was successfully created, it creates the dialog window, sizing it based on the size of the form. Again it checks for success, and assuming that worked, it calls a function to initialize the form for a specific color value (the starting color). That color value is contained in a constant called `iSTARTCOLOR`. Finally, the form is selected into the dialog window, the dialog window is centered on the display (since it has no parent), and it then is set to wait forever (or until the dialog is made invisible, either by calling the `setvisible()` method, or by the user clicking the Close gadget.

The program as is will not yet compile without warnings nor will it run without errors. The source code of the form contains assignments of function names for handling events, but those functions have not yet been created.

Getting the Basic Form Running

In this section we will now add all the minimum bits remaining in order to get our program to run and show the form. It won't do anything yet, but at least we will be able to see the form come up in the dialog window, and we will also be able to close the window and have the program exit correctly.

In order to do this, we need to at least add the functions, even if they don't yet do anything. That code is shown below.

Example 5.2. The remaining Empty Functions of the colorlab Program

```
function adjustformcolorvals(wxform f, integer rgbval, \
                             string ignorescrollbar="")
end function

function hexval_olf(wxformedittext me)
end function

function decval_olf(wxformedittext me)
end function

function redval_olf(wxformedittext me)
end function

function greenval_olf(wxformedittext me)
end function

function blueval_olf(wxformedittext me)
end function
```

```
function redscroll_os(wxformscrollbar me, string scrolltype)
end function

function greenscroll_os(wxformscrollbar me, string scrolltype)
end function

function bluescroll_os(wxformscrollbar me, string scrolltype)
end function

function okbtn_oc(wxformbutton me)
  wxdialog d

  d =@ me.form.container
  d.setvisible(.false)
end function

function errormsg(string s)
  wxmessagedialog(message=s, captiontext="SIMPOL Color Lab \
                  Error", style="ok", icon="error")
end function
```

The remaining functions shown above don't yet do anything, except for the `okbtn_oc`, which merely sets the dialog visibility to `.false`, which results in the code exiting the `processmodal()` method and the program then exits.

Note

Another useful naming convention can be seen in the names of the functions. The beginning portion of the name indicates the control with which it is associated, followed by an underscore, and then a set of letters describing the event. Below is a table of potentially useful names for the portion following the underscore.

Table 5.2. Event Handling Function Naming Conventions

Suffix	Explanation
oc	onchange
oc	onclick
occ	oncellchange
ocs	oncellselect
ocwc	oncolwidthchange
odc	ondoubleclick
ogf	ongotfocus
olf	onlostfocus
om	onmouse
om	onmove
orhc	onrowheightchange
os	onscroll
os	onselect
os	onsize
osc	onselectionchange
osc	onstatechange

Suffix	Explanation
ovc	onvisibilitychange

At this point, it should be possible to build and run the program. It doesn't do anything yet, other than display the form, but it is a nice place to be, since now all that is left is manipulating the form in response to user-generated events (or in other words, filling in those empty functions).

Finishing the Color Lab Program

Now that we have the basic program running and displaying the form, all that remains is to fill in the functions that are currently empty. One thing that we can do to minimize the amount of coding is to use a common piece of code for some of the functions, and call it from each of them. From looking at the code, it seems that the scrollbar event handling functions will probably be similar, as will the functions that handle the edit control events for the three color values. Everything will eventually call the `adjustformcolorvals()` function. Since that is the function that everything has in common (it is even being called during initialization), let's build that first.

Example 5.3. The Full Implementation of the `adjustformcolorvals()` Function

```
function adjustformcolorvals(wxform f, integer rgbval, \
                             string ignorescrollbar="")
  integer red, green, blue

  blue = rgbval / 0x10000
  green = (rgbval mod 0x10000) / 0x100
  red = ((rgbval mod 0x10000) mod 0x100)
  f!tbHexColorValue.settext(.tostr(rgbval, 16))
  f!tbDecColorValue.settext(.tostr(rgbval, 10))
  f!tbRed.settext(.tostr(red, 10))
  f!tbGreen.settext(.tostr(green, 10))
  f!tbBlue.settext(.tostr(blue, 10))
  f!sbRed.setbackgroundrgb(red)
  f!sbGreen.setbackgroundrgb(green * 0x100)
  f!sbBlue.setbackgroundrgb(blue * 0x10000)
  if ignorescrollbar != "red"
    f!sbRed.setscroll(position=red)
  end if
  if ignorescrollbar != "green"
    f!sbGreen.setscroll(position=green)
  end if
  if ignorescrollbar != "blue"
    f!sbBlue.setscroll(position=blue)
  end if
  f!rBorder.setrgb(rgb=rgbval)
end function
```

What this function does, is to take the final color value and then use it to set the value of all the other controls. While playing around with this, it was noticed that setting the position of a scrollbar that had caused the event resulted in a strange flicker, so an extra parameter was created that is ignored by the other functions, but which is passed by the code that handles the scrollbar events. That let's the function choose not to set the scroll position for the scrollbar that is passed. Other than that, the function is fairly basic. It takes the color value that comes as an RGB value and splits it into the red, green, and blue values. It then assigns the value to each of the edit controls. It also uses the individual color values to set the background color for each of the scrollbars, as well as being used to set the position of the thumb in the scrollbars. Finally, it sets the color of the rectangle to that of the color passed.

The next two functions are quite similar, but different enough to deserve different function implementations. In each case, the functions retrieve the current value of the control, convert it to a value, force the value to be within a valid range, and then they each call the `adjustformcolorvals()` function.

Example 5.4. The Code for the `hexval_olf()` and `decval_olf()` Functions

```
function hexval_olf(wxformedittext me)
  integer rgbval
  string hexcolor

  hexcolor = me.gettext()
  hexcolor = .if(hexcolor <= "", "0", hexcolor)

  rgbval = .toval(hexcolor, nohexdigits(hexcolor), 16)
  rgbval = .max(.min(0xffffff, rgbval), 0)
  adjustformcolorvals(me.form, rgbval)
end function

function decval_olf(wxformedittext me)
  integer rgbval
  string deccolor

  deccolor = me.gettext()
  deccolor = .if(deccolor <= "", "0", deccolor)

  rgbval = .toval(deccolor, nodigits(deccolor), 10)
  rgbval = .max(.min(0xffffff, rgbval), 0)
  adjustformcolorvals(me.form, rgbval)
end function
```

There are two special function calls in the previous code, `nohexdigits()` and `nodigits()`. Each is designed to extract all of the characters of a specific type, either normal digits or the normal plus hexadecimal digits. The result is passed to the `.toval()` function as the characters to ignore when converting the value.

The next task is to handle the events for the individual color values. As mentioned earlier, these will, in fact, be exactly the same code in each case, since the change to any one color value still requires all the color values to be read. All the event handlers will call the exact same function, which we will call `handleonecolorchange()`.

Example 5.5. Handling the Events for the Color Edit Controls

```
function handleonecolorchange(wxform f)
  integer red, green, blue, rgbval
  string color

  color = f!tbRed.gettext()
  red = .toval(color, nodigits(color), 10)
  red = .max(0, .min(255, red))
  color = f!tbGreen.gettext()
  green = .toval(color, nodigits(color), 10)
  green = .max(0, .min(255, green))
  color = f!tbBlue.gettext()
  blue = .toval(color, nodigits(color), 10)
  blue = .max(0, .min(255, blue))
  rgbval = calcrgbval(red, green, blue)
  adjustformcolorvals(f, rgbval)
end function

function redval_olf(wxformedittext me)
  handleonecolorchange(me.form)
end function

function greenval_olf(wxformedittext me)
  handleonecolorchange(me.form)
end function

function blueval_olf(wxformedittext me)
  handleonecolorchange(me.form)
end function

function calcrgbval(integer red, integer green, integer blue)
  integer rgbval

  rgbval = blue * 0x10000 + green * 0x100 + red
end function rgbval
```

The final piece of the puzzle is to handle the events for the scrollbars, and this next piece of code does that. Again, all three have much in common, so they all call one common routine called `doscrollbars()`.

Example 5.6. Handling the Scroll Bar Events

```

function getcurrentcolorvals(wxform f, integer red, \
                             integer green, integer blue)
  red = f!sbRed.position
  green = f!sbGreen.position
  blue = f!sbBlue.position
end function

function doscrollbars(wxform f, string ignorescrollbar)
  integer red, green, blue, rgbval

  red = 0; green = 0; blue = 0
  getcurrentcolorvals(f, red, green, blue)
  rgbval = calcrgbval(red, green, blue)
  adjustformcolorvals(f, rgbval, ignorescrollbar)
end function

function redscroll_os(wxformscrollbar me, string scrolltype)
  doscrollbars(me.form, "red")
end function

function greenscroll_os(wxformscrollbar me, string scrolltype)
  doscrollbars(me.form, "green")
end function

function bluescroll_os(wxformscrollbar me, string scrolltype)
```

```
  doscrollbars(me.form, "blue")
end function
```

The first function in the previous chunk of code is called by the `doscrollbars()` function to retrieve the component color values from the position property of each of the scrollbars. Since we set the range of the scrollbars to 256 and the thumb size to 1, that means that the range of valid positions is from 0 through 255. Once the component values have been retrieved, it calls the `calcrgbval()` function that is also called by the `handleonecolorchange()` function.

Finally, here is the code for the two functions mentioned earlier for extracting the valid digits from the string passed.

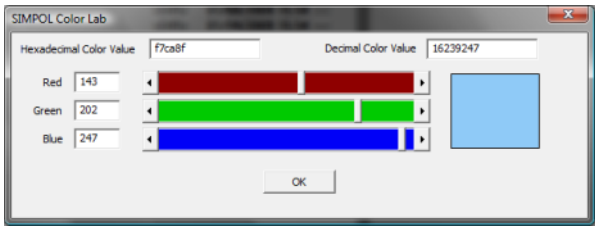
Example 5.7. Extracting the Digits from String Values

```
function nodigits(string s)
end function s-"0"-"1"-"2"-"3"-"4"-"5"-"6"-"7"-"8"-"9"

function nohexdigits(string s)
end function s-"0"-"1"-"2"-"3"-"4"-"5"-"6"-\
               "7"-"8"-"9"-"a"-"b"-"c"-"d"-\
               "e"-"f"-"A"-"B"-"C"-"D"-"E"-"F"
```

Summary

In the preceding section we have learned to design a basic form using the Form Designer, including setting default values, using system theme colors, and working with the sizing and alignment tools. We have also saved that form in the new XML-based form format and also as SIMPOL source code. We have worked with included source files and SIMPOL language libraries. Finally, the resulting form was incorporated in a project that used a dialog to host the form and then waited on events, which were then handled by the program code. We also validated the input that was entered.

This sort of program component is a common requirement for more complex applications, where any number of similar dialog-style user-interface components will be needed to ensure a user-friendly experience. This sort of program is a stepping stone to the type of project discussed in the next chapter, which is considerably more complex.



Hexadecimal Color Value: f7ca8f
Decimal Color Value: 16239247
Red: 143
Green: 202
Blue: 247

The final working SIMPOL Color Lab program.

Chapter 6. GUI-Style Database Programs

Introduction

Database programs with a graphical user interface (GUI) are a very common type of application. In many cases, this is the only type of application that some people may need to create, though if the package becomes complex, it may well use quite a number of dialogs to get input from users. In this chapter, we will cover the basics of building a database-based graphical program to provide a starting point for building more complex systems of this nature.

Tip

Before reading and getting heavily involved in this chapter, it is a good idea to at least read through Chapter 5, *Dialog-Style Programs*. Many of the techniques for working with the Form Designer are covered in that chapter.

In this chapter we will build a basic contact manager. This will be a program that manages an address database using a form hosted in a window. We will also use a tool bar for selecting records, and a menu with a small set of items. The steps to follow in creating this application are:

1. Create the database
2. Create the form
3. Build the application code

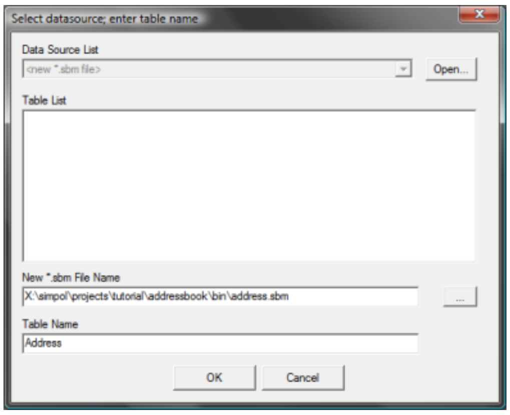
In essence, this is not very dissimilar to the steps followed to create the dialog-style application, except first we need to create the database. So, let's get started!

Creating the Project

Just as we did in Chapter 5, *Dialog-Style Programs*, it would be a good idea to create the project first, since that will provide us with a location for storing the components of our project that we are creating using Superbase NG Personal. Start the SIMPOL IDE and once it is running from the menu, select File → New Project. In the dialog window that appears, select an appropriate location for the project, and give it the name `addressbook`. Then click on the OK button to create the project. Take note of this location, we will use it save our new database in a moment, plus our form design later. For now, minimize or close the IDE;.

Create the Database

In many cases, a database is a complex set of related tables, each containing a specific set of information and also fields which link to other tables. In this example, we only really need one table for now, the "`Address`" table. To create this table, we need to start up Superbase NG Personal. Once it is running, select File → New → Table…. That will display the data source dialog. Since no data sources are currently open, the combo box with the list of data sources will be disabled and the list of tables will be empty. In the New *.sbm File Name box enter the location of your project's `bin` directory, plus `address.sbm` and in the Table Name box enter `Address`.



Data Source List: <new *.sbm file>

New *.sbm File Name: X:\simpol\projects\tutorial\addressbook\bin\address.sbm

Table Name: Address

The Superbase NG Personal New Database dialog.

Now click on the OK button to create the database container and to display the dialog window where we can define the table.

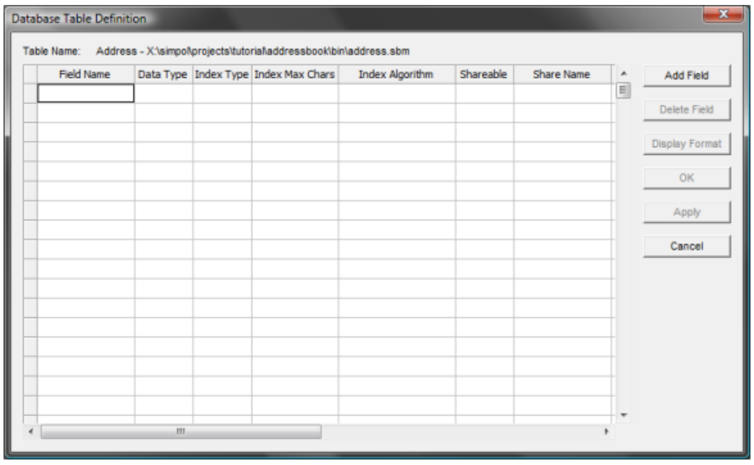


Table Name: Address - X:\simpol\projects\tutorial\addressbook\bin\address.sbm

Field Name	Data Type	Index Type	Index Max Chars	Index Algorithm	Shareable	Share Name

The Superbase NG Personal Database Table Definition dialog.

To add a field, just click on the Add Field button. The first field we will add is the `AddressID` field. Enter that text into the box and click on the OK button. The screen will look like the one below:

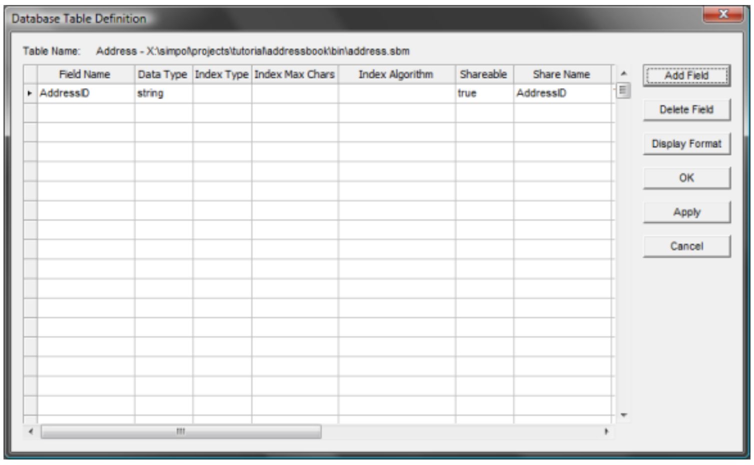


Table Name: Address - X:\simpol\projects\tutorial\addressbook\bin\address.sbm

Field Name	Data Type	Index Type	Index Max Chars	Index Algorithm	Shareable	Share Name
AddressID	string				true	AddressID

The updated Superbase NG Personal Database Table Definition dialog.

The default data type for a new field in an empty table is string (text). We will change this after we have added a number of other fields, since the fields inherit the data type and other characteristics from the currently selected field. For now add the next field, `FirstNames`, which will contain the first and any middle names or initials. Now move the scrollbar at the bottom to the right, to expose some more columns. The three columns named Shareable, Share Name, and Share Type are specific to using the multi-user database support, and can be ignored for now. The default assignments tend to be adequate. The Display Format column is used for more than one purpose. It is used as part of the multi-user database engine, but it is also used to supply the desired display format in the data-aware form environment. It is currently set to 4000 (which happens to be the maximum length of a text field that can be accessed via the PPCS protocol used for the current multi-user support. Change that to `30`. This column will be the only one we need to change as we go through them later, but by changing this one now, we may have less to change later. See the image that follows.

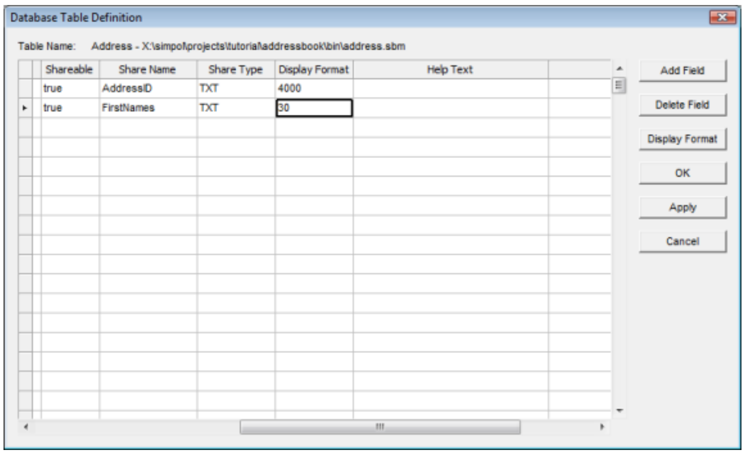


Table Name: Address - X:\simpol\projects\tutorial\addressbook\bin\address.sbm

Shareable	Share Name	Share Type	Display Format	Help Text
true	AddressID	TXT	4000	
true	FirstNames	TXT	30	

The Display Format column in the Database Table Definition dialog.

Note

The SBME database format does not actually have any limitations on the size of a text column, or an integer, etc., but since the initial multi-user support was designed to be 100% compatible with Superbase's PPCS protocol, it has the exact same limitations as well. A later version of the protocol that is not intended to be Superbase compatible is planned, and will support all SIMPOL data types in their full capabilities. Also, the database engine only stores what is there, it uses variable length fields to only take up the space it must.

Now add the rest of the fields, if the currently selected field is the `FirstNames` field, then the length of `30` will also be inherited. When you get to the `Address1` field, change the length to 50, and then make sure that is selected when adding the rest.

- Surname
- Address1
- Address2
- Address3
- Address4
- City
- Region
- Postcode
- CountryCode
- Telephone

- Fax
- Email
- Remarks

Now that that is finished, go through and change the field lengths to 25 for `Postcode`, `Telephone`, and `Fax`, 80 for `Email`, and 2 for `CountryCode`. Now set the `Remarks` field to `4000 R`, which allows for new line characters within the field content. The Display Format column should look like this:

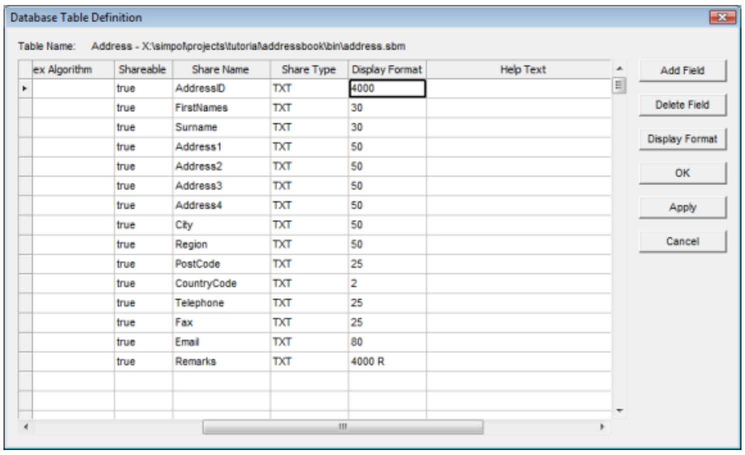


Table Name: Address - X:\simpol\projects\tutorial\addressbook\bin\address.sbm

ex Algorithm	Shareable	Share Name	Share Type	Display Format	Help Text
	true	AddressID	TXT	4000	
	true	FirstNames	TXT	30	
	true	Surname	TXT	30	
	true	Address1	TXT	50	
	true	Address2	TXT	50	
	true	Address3	TXT	50	
	true	Address4	TXT	50	
	true	City	TXT	50	
	true	Region	TXT	50	
	true	PostCode	TXT	25	
	true	CountryCode	TXT	2	
	true	Telephone	TXT	25	
	true	Fax	TXT	25	
	true	Email	TXT	80	
	true	Remarks	TXT	4000 R	

The Display Format column after updating the format entries.

Now scroll back to the left and select the cell for the `AddressID` Data Type column. Click again to drop down the selection box, as shown below, and pick `integer`. Then pick the cell in the next column, Index Type, and click again to drop down the selection box, and pick `unique` as the index type, then click on the next cell to see the results. It should look like the following:

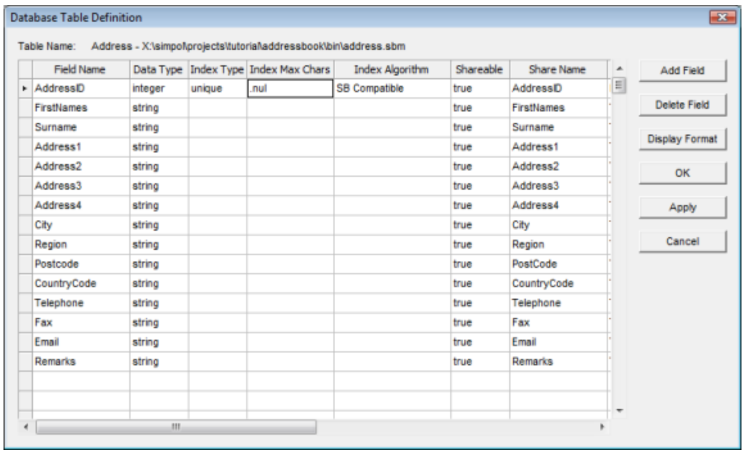
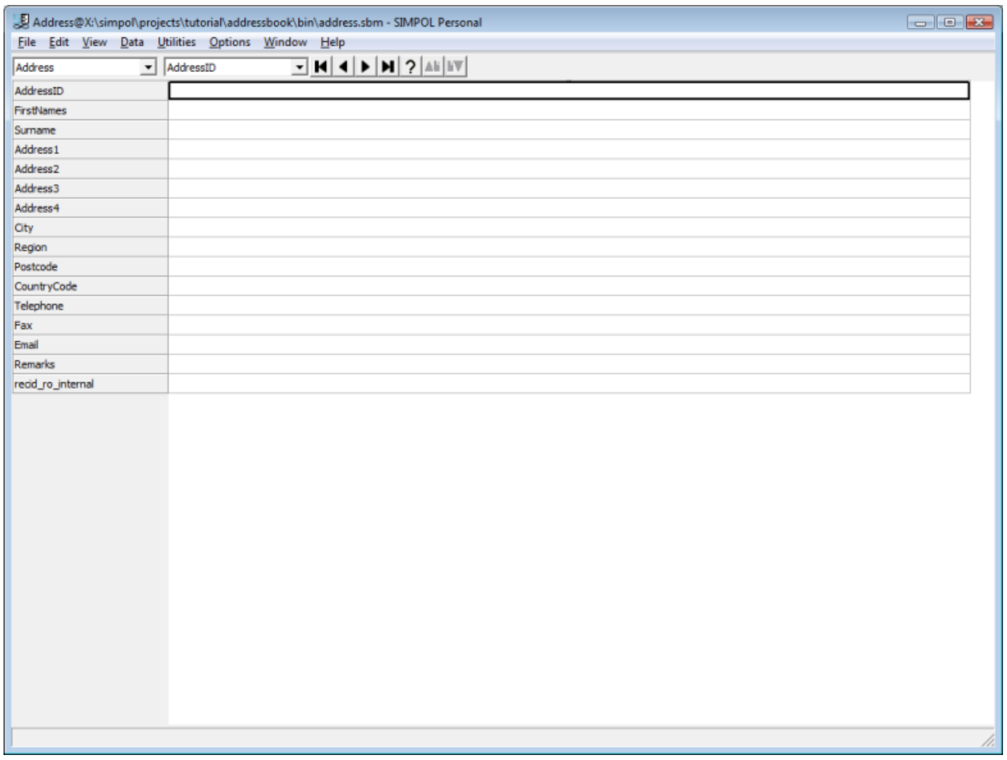


Table Name: Address - X:\simpol\projects\tutorial\addressbook\bin\address.sbm

Field Name	Data Type	Index Type	Index Max Chars	Index Algorithm	Shareable	Share Name
AddressID	integer	unique	.nul	SB Compatible	true	AddressID
FirstNames	string				true	FirstNames
Surname	string				true	Surname
Address1	string				true	Address1
Address2	string				true	Address2
Address3	string				true	Address3
Address4	string				true	Address4
City	string				true	City
Region	string				true	Region
Postcode	string				true	PostCode
CountryCode	string				true	CountryCode
Telephone	string				true	Telephone
Fax	string				true	Fax
Email	string				true	Email
Remarks	string				true	Remarks

The `AddressID` entry after changing the data type and adding an index.

Finally, change the display format from the huge list of 9s to something a little more friendly and clear, such as six 0's, either directly or via the Display Format button. Now click on the OK button to save the table definition off. The final result should now be shown in Superbase NG Personal in Record View, as shown here:

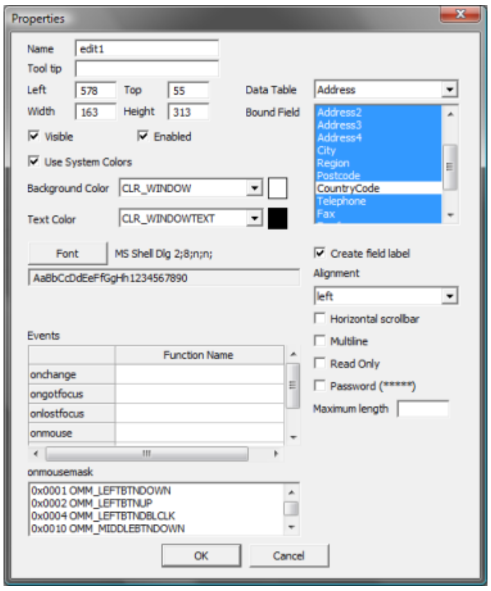


Superbase NG Personal after saving the table definition for the `Address` table.

Building the Form

Now that our basic database table has been created, it is time to create a form so that we can manage the data effectively and for creating our application. Select File → New → Form… from the menu to create a new form. Then double-click the left mouse button and select the Use system colors check box and click on OK.

This time, we are going to add a number of controls in one step. Select the edit control tool from the tool bar, or choose Draw → Editable Text from the menu, and then starting roughly in the center of the form horizontally and near the top vertically, drag a rectangle and then let go. In the Properties dialog, select the Data Table combo box and from there, select the `Address` table entry. Then drag select the entries from `FirstNames` through `Postcode`. Then while holding the **Ctrl** key depressed, also select the entries: `Telephone`, `Fax`, `Email`, and `Remarks`. When all are selected, click on the check box below the field list entitled: Create field label. The window should look something like this:


Properties

Name: edit1
Tool tip:
Left: 578 Top: 55
Width: 163 Height: 313
☑ Visible ☑ Enabled
☑ Use System Colors
Background Color: CLR_WINDOW
Text Color: CLR_WINDOWTEXT
Font: MS Shell Dlg 2;8;n;n;
AaBbCcDdEeFfGgHh1234567890

Data Table: Address
Bound Field: Address2, Address3, Address4, City, Region, Postcode, CountryCode, Telephone, Fax
☑ Create field label
Alignment: left
☐ Horizontal scrollbar
☐ Multiline
☐ Read Only
☐ Password (*****)
Maximum length:

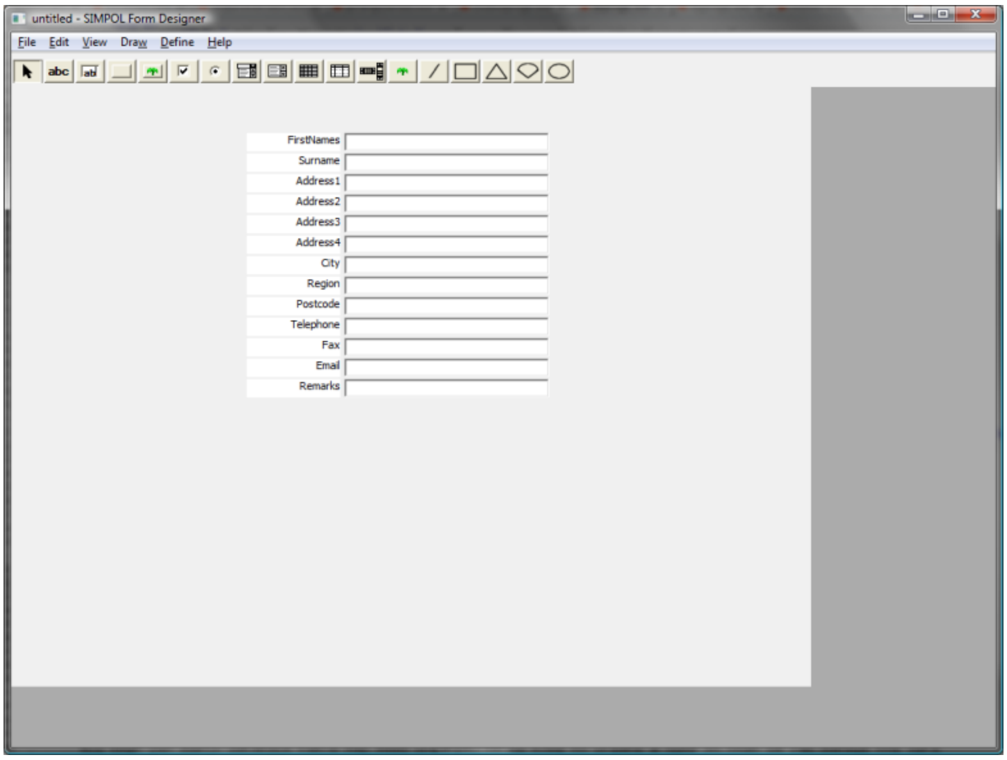
Events

	Function Name
onchange	
ongotfocus	
onlostfocus	
onmouse	

onmousemask
0x0001 OMM_LEFTBTNDOWN
0x0002 OMM_LEFTBTNUP
0x0004 OMM_LEFTBTNDBLCLK
0x0010 OMM_MIDDLEBTNDOWN

The Properties dialog selecting multiple fields for the edit control.

Now click on the OK button. On the form, a text control for the label and an edit control for the content will be created for each selected field in the list. They are all created with the same foreground and background colors, so the text controls will need to be adjusted.

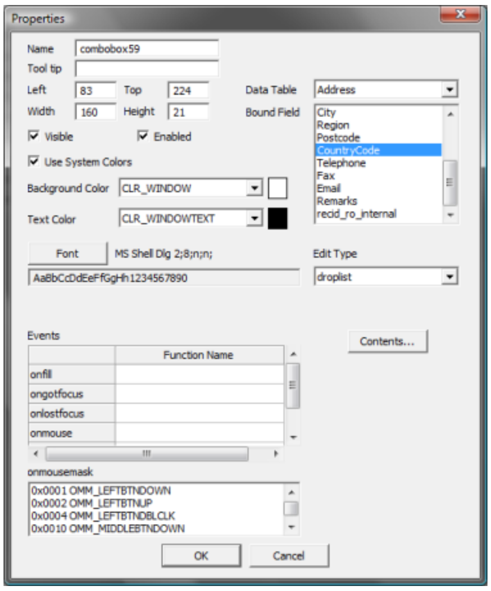

untitled - SIMPOL Form Designer

FirstNames
Surname
Address1
Address2
Address3
Address4
City
Region
Postcode
Telephone
Fax
Email
Remarks

The form after adding multiple controls at once.

To resolve this, click and drag a rectangle around only the label controls. Once all have been selected, click with the right button and from the menu select the Graphic Properties item. In the dialog window, change the Background Color entry to `CLR_BTNFACE` and then click on the Set Back Color button. Now change the Text Color entry to `CLR_BTNTEXT` and click on the Set Text Color button. Click on either the Cancel or the Close gadget to exit.

Now we can position the content, and add the remaining fields. The `AddressID` field content is meant to be created programmatically, so it should be added as a bound text control instead of an edit control. Create the label and content as separate steps. Move the labels and edit controls into position by selecting the group and then grabbing the widget at the center (it may not be visible, but the cursor will change appropriately). For the `CountryCode` field add a combo box control. In the dialog, select the `CountryCode` field from the Bound Field list.



Properties

Name: combobox59
Tool tip:
Left: 83 Top: 224
Width: 160 Height: 21
☑ Visible ☑ Enabled
☑ Use System Colors
Background Color: CLR_WINDOW
Text Color: CLR_WINDOWTEXT
Font: MS Shell Dlg 2;8;n;n;
AaBbCcDdEeFfGgHh1234567890
Data Table: Address
Bound Field: City, Region, Postcode, CountryCode, Telephone, Fax, Email, Remarks, recid_ro_internal
Edit Type: droplist

Events

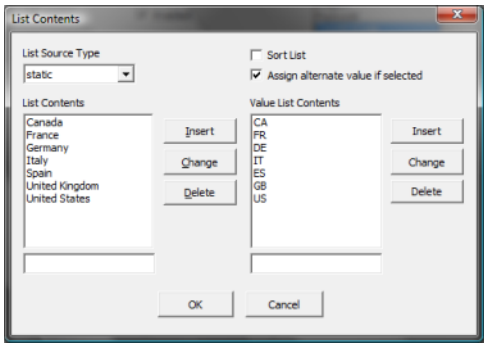
	Function Name
onfill	
ongotfocus	
onlostfocus	
onmouse	

onmousemask

0x0001 OMM_LEFTBTNDOWN
0x0002 OMM_LEFTBTNUP
0x0004 OMM_LEFTBTNDBLCLK
0x0010 OMM_MIDDLEBTNDOWN

The Properties dialog for the combo box control.

Now click on the Contents… button. In the resulting dialog window, from the List Source Type combo box, select the `static` entry. Now in the blank edit control at the bottom of the list, add the value `Canada`, then click on the Insert button. Continue with the values: `France`, `Germany`, `Italy`, `Spain`, `United Kingdom`, and `United States`. Also click on the check box entitled: Assign alternate value if selected. Now in the Value List Contents section, add the following values in the same way as before: `CA`, `FR`, `DE`, `IT`, `ES`, `GB`, and `US`. The dialog window should look similar to this:



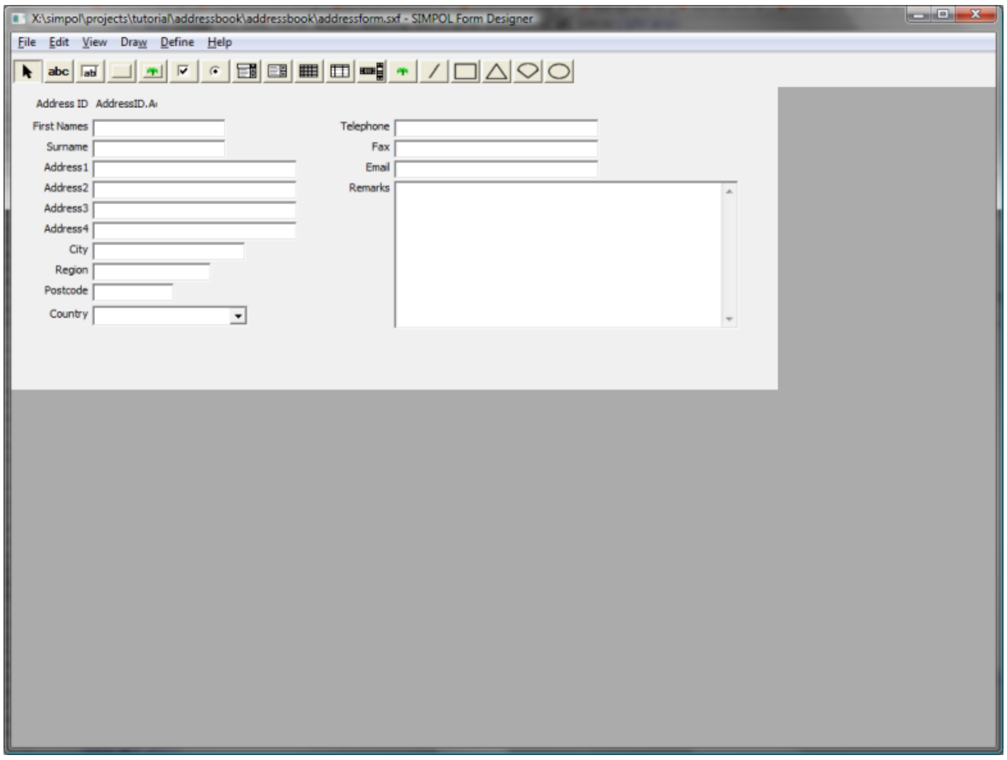
List Contents

List Source Type: static
☐ Sort List
☑ Assign alternate value if selected

List Contents	Value List Contents
Canada	CA
France	FR
Germany	DE
Italy	IT
Spain	ES
United Kingdom	GB
United States	US

The List Contents dialog for the combo box control.

The only change remaining is to size the `Remarks` control larger, and then double-click on it and tick the Multiline check box so that formatted text can be added into the control. After rearranging the controls, and resizing the form (assigning the name addressform to both form and page names), the screen looks like this:


X:\simpol\projects\tutorial\addressbook\addressbook\addressform.sxf - SIMPOL Form Designer
File Edit View Draw Define Help
Address ID AddressID.A
First Names
Surname
Address1
Address2
Address3
Address4
City
Region
Postcode
Country
Telephone
Fax
Email
Remarks

The final look of the form.

To make things a bit more friendly, we will want to put the focus into the first field when a new record is created, and to make that easier, we should give the dataform1edit control a more useful name, so double-click on that and change the control name to `tbFirstnames`. Now click the OK button.

The final bit of tweaking is to modify the tab order. Every single control is part of the tab order, since in SIMPOL, the tab order and the z-order are the same. Click on the Define → Tab Sequence item and the list of controls will be shown with their names. Now we haven't bothered to assign special names to the controls this time, so the names won't be terribly meaningful, but as the controls are selected in the list, a colored border is placed around each item on the form. Multiple controls can be selected at once, and moved as a block up or down. Use this tool to arrange the controls in the order desired. Any changes are not permanent until the OK button is clicked. Save the form as a form (not a program) into our project source directory, which is the directory of the same name as the project below the root project directory. So if the project is called `AddressBook`, then it will be in a directory called `AddressBook` and that will have two subdirectories, `bin` and `AddressBook`. The second of these is the source directory.

Well, so far so good. The form has been created and we are ready to start diving into the code, which we will do in the next section.

The Program Code

The good news is that there isn't actually much program code to write. In fact, this example is provided in the `Projects\tutorial\addressbook` directory, as are the other samples from this book if they are not found elsewhere. The other good news is that if you were building a different application, you would still have very little coding to do, since the pieces that make up the `addressbook` project can be used as the basis of *any* database-based GUI project.

In this chapter, we won't go into all the program code, instead we will work with the main pieces that are affected. For the full story, there is no substitute for opening the actual project and reading through the source and the comments there.

The `main()` Function

The `main()` function of the program is where the code execution begins. When it exits this function the program should normally end (unless there are still separate threads running that have not yet exited).

Example 6.1. The `main()` function of the program

```
function main()
  addressbookapplication app
  appwindow appw
  string cd, formfilename, dirsep
  integer e
  wxmenubar mb
  wxtoolbar tb
  wxstatusbar sb
  point lt, br
  syscolors colors

  dirsep = getdirectorysepchar()
  cd = getcurrentdirectory()
  formfilename = cd + dirsep + "addressform.sxf"
  colors =@ syscolors.new()

  e = 0
  mb =@ mainmenu()
  if mb !@= .nul
    sb =@ wxstatusbar.new(error=e)
    if sb !@= .nul
      tb =@ buildiconbar(colors)
      if tb !@= .nul
        app =@ addressbookapplication.new(mb, tb, sb, "", "")
        if app !@= .nul
          appw =@ app.windows.getfirst()
          if not fileexists(formfilename)
            wxmessagedialog(appw.w, "Form file 'addressform.sxf' \
                            not found", sAPPMSGTITLE, "ok", \
                            "error")
          else
            appw.openformdirect(formfilename, sAPPMSGTITLE)
            if appw.form !@= .nul
              prepaddressbookform(appw)
              appw.resizewindowtoform()

              lt =@ point.new(0, 0)
              br =@ point.new(0, 0)
              getcenteredwindowrect(appw.outerwidth, \
                                    appw.outerheight, lt, br, \
                                    error=e)
              if e == 0
                appw.setposition(lt.x, lt.y)
              end if
              appw.setcurrentpath(cd)
              selectrecord(appw, "selectcurrent", silent=.true)
              appw.w.setstate(visible=.true)
              app.run()
            end if
          end if
          app.exit()
        end if
      end if
    end if
  end if
end function
```

Starting from the top, we declare a variable of type addressbookapplication (more on that later), plus a variable of type appwindow. The appwindow type is provided by the `appframework.sml` library. The other variables should be relatively self-explanatory: menu bar, tool bar, and status bar. The point is used for centering the window on the display.

After initializing the path name for the form file, the program attempts to create the menu bar, the status bar, and the tool bar. If all of those are created successfully (and they should be), the `app` variable is assigned the newly created addressbookapplication object. Assuming it was created successfully, we retrieve the first (and currently only) appwindow object and open the form file we created earlier. Assuming that worked correctly, we prepare the form (by assigning certain event handlers), then we resize the window to fit the form, center the window on the display, and reselect the current record (which helps if there are form-based calculations that need to be recalculated now that the event handlers might be in place. Finally, we enter the `run()` method of the addressbookapplication object.

The addressbookapplication Type

In the design of this program, a key component is the addressbookapplication type. So let's look at it:

Example 6.2. The addressbookapplication type

```
type addressbookapplication (application)
  reference
  application __app resolve
  type(db1table) address
end type
```

That doesn't really explain much, but that is because this is an enhancement to the application type that is supplied by the `appframework.sml` library. Let's have a look at that one now too:

Example 6.3. The application type

```
type application (application) export
  embed
  string title
  dring windows
  dring datasources
  boolean running
  string inifilename
  integer ostype
  event onexitrequest

  reference
  type(*) _
  type(*) __ resolve

  wxbitmap windowicon
  ppcstype1 ppcs
  sysinfo systeminfo
  localeinfoold SBLlocale
  localeinfo locale
  tdisplayformats displayformats
  function run
  function exit
  function adddatasource
  function closedatasource
  function datasourceunused
  function finddatasrc
  function opendatasource
end type
```

There, that's a little more meaty. On closer examination we can discover the `run()` method listed in the type. That is the main loop for the application framework. The program sits in that function all the time waiting for events. The `exit()` method is not used. The rest are used to open, find, and close data sources. As long as the running property is set to `.true`, the program will remain in the main loop in the `run()` method.

Note

So why did we bother to create our own type, why not just use the application type as it is? In the current example it wasn't absolutely necessary. However, it turns out that it is useful. When we created the table we also set up one field as a unique index, and we will need a way of creating that value (SIMPOL does not currently do that for us). During the function that will handle the onnewrecord event, easy access to the `address` table will make the code easier to write.

Therefore, we created our own type, placed an application property into it and made it **reference** (so that we have to initialize it), and **resolve** (so that its properties resolve as properties of the addressbookapplication object). Since we declared the application type as **resolve**, we can also declare the addressbookapplication type to have a type tag of `application`. This allows variables to be declared like this: `type(application)`, which means they can contain any variable that is tagged with the `application` tag. Good design dictates that we should then make sure that anything tagged this way can be used as if it were the application type.

Now we should look at the most significant function here, the `new()` method of the addressbookapplication type. That is where the majority of the initialization takes place.

Example 6.4. The Code to Create a New addressbookapplication

```
function addressbookapplication.new(addressbookapplication me, \
                                    wxmenubar mb, wxtoolbar tb, \
                                    wxstatusbar sb, \
                                    string iconname, \
                                    string iconimagetype)
  appwindow appw
  datasourceinfo src
  type(db1table) t
  integer e
  boolean ok

  ok = .false
  e = 0
  me.__app =@ application.new(appiconfile=makenotnull(iconname), \
                              iconimagetype=\
                              makenotnull(iconimagetype), \
                              inifilename="", apptitle=sAPPTITLE)
  me.__app.__ =@ me
  me.onexitrequest.function =@ exit
  me.running = .true
  appw =@ appwindow.new(me, visible=.false, mb=mb, tb=tb, sb=sb)
  if appw =@= .nul
    wxmessagedialog(message="Error creating window", captiontext=\
                    sAPPMSGTITLE, style="ok", icon="error")
  else
    initmainmenu(appw.mb, me)
    appw.onmanagemenu.function =@ managemenu
    inittoolbar(appw.tb, appw)
    appw.onmanagetoolbar.function =@ managetoolbar

    src =@ me.opendatasource("sbme1", "address.sbm", appw, error=e)
    if src =@= .nul
      wxmessagedialog(appw.w, "Error opening the address.sbm \
                      file", sAPPMSGTITLE, "ok", "error")
    else
      t =@ appw.opendatatable(src, "Address", error=e)
      if t =@= .nul
        wxmessagedialog(appw.w, "Error opening the 'Address' \
                        table", sAPPMSGTITLE, "ok", "error")
      else
        me.address =@ t
        ok = .true
      end if
    end if
  end if

  if not ok
    me =@ .nul
  end if
end function me
```

Starting from the top, the first thing the code does is create a new application object and assign the reference to that object to the me.__app property. That ensures that all of the properties and methods of the application object are also available as part of the addressbookapplication type. The next rather arcane looking bit is the assignment of a reference to the `me` variable to the __ (double underscore) property of the application object that we just created. This somewhat circular reference is quite important, since it means that all of the properties of the wrapper addressbookapplication object are also available to the application object.

That is a bit convoluted, but in practice it is fairly easy and powerful. To understand it, it helps to understand the problem it solves. When an event occurs that is associated with the application object, only the application object is passed to the event handling function. If the function needs access to the wrapper object, it needs a way to get to that. Although it would be possible to pass the wrapper object as the optional reference parameter, that may be needed for something else. By assigning a reference to the wrapper object to the underscore or double-underscore property, the function can have full access to the capabilities of the wrapper object.

Tip

The single and double underscore properties are part of most SIMPOL complex data types. They were added to allow the user to add their own information to an existing type. Both properties are **reference** properties (they refer to an object), but the double underscore property is also marked as **resolve**, which means that whatever object is assigned here will take part in the resolution of the dot operator. What that means in practice is that a variable called `app` that refers to the application object portion of the addressbookapplication object, will still be able to reach the address property of the addressbookapplication. Please note that the IDE will not be able to show this, since it happens at run time.

Returning to our initialization code, we assign a function to handle the onexitrequest event, which will be called if there are no more visible windows (this is part of the application object). The running property is set to `.true` (setting this to `.false` will cause the program to initiate shutdown), and then the initial window of the program is created. To that we pass the menu bar, tool bar, and status bar objects that we created earlier in the program code. We are creating the window invisibly, since we won't show it until later once the form has been loaded.

Once we have successfully created the initial window, we then initialize the menu and tool bars, and assign a function to handle the onmanagemenu and onmanagetoolbar events of the appwindow object. These are called whenever something has been done that might warrant a change to the menu or tool bar state, such as opening a form, creating a new record, closing a table, etc.

Finally we open the data source (our `address.sbm` file) and the data table (`Address`). The first is opened via a method of the application object, since data sources are managed at the application level, and the table is opened by the appwindow object, since tables are managed at the window level (the framework is designed to allow each window to open its own table objects). Finally we assign the table to the property that we defined for it in our wrapper type; the remainder of the function is self-explanatory.

The Remaining Initialization Code

The rest of the program code is mainly the definition of the menu and tool bars, plus the code to handle the events that have been defined. We will look briefly at the code that creates and initializes the menu and tool bars.

Example 6.5. The Code for the Menu Bar

```
function mainmenu()
  wxmenubar mb

  mb =@ wxmenubar.new()
  // This section creates the File menu.
  wxmenu filemenu
  filemenu =@ wxmenu.new()
  filemenu.insert("","E&xit", name="exit")

  // This section creates the Data menu.
  wxmenu datamenu
  datamenu =@ wxmenu.new()
  datamenu.insert("","&Add{9}Ctrl+N", name="add")
  datamenu.insert("","&Save{9}Ctrl+S", name="save")
  datamenu.insert("","&Delete{9}Ctrl+Del", name="delete")

  // This section creates the Help menu.
  wxmenu helpmenu
  helpmenu =@ wxmenu.new()
  helpmenu.insert("","&About " + sAPPTITLE + "...", name="about")


  mb.insert(filemenu, "&File", name="file")
  mb.insert(datamenu, "&Data", name="data")
  mb.insert(helpmenu, "&Help", name="help")
```

```
end function mb
```

Creating a menu bar is not particularly complicated, as we can see here. In this particular case, the definition of the functionality for handling the events when a menu item is selected has not yet been included. This is deliberate, since it allows us to create the menu bar before the window even exists. Later, when the window has been created, we will call the `initmenubar()` to add the handlers for the events, plus the reference object for each event.

The code here should be fairly clear. We create an empty wxmenubar object. Then we create the top level wxmenu objects and proceed to fill these with entries. Once all the top-level menus have been created, they are added to the menu bar. Finally, the function returns the newly-created menu bar object as its return value.

Now that the menu bar has been created, let's look at the code to initialize it.

Example 6.6. The Code for the Menu Bar

```
function initmainmenu(wxmenubar mb, addressbookapplication app)
  mb!file.menu!exit.onselect.function =@ exitviamenu
  mb!file.menu!exit.onselect.reference =@ app

  mb!data.menu!add.onselect.function =@ newrecord
  mb!data.menu!add.onselect.reference =@ app
  mb!data.menu!save.onselect.function =@ saverecord
  mb!data.menu!save.onselect.reference =@ app

  mb!data.menu!delete.onselect.function =@ deleterecord
  mb!data.menu!delete.onselect.reference =@ app

  mb!help.menu!about.onselect.function =@ helpabout
  mb!help.menu!about.onselect.reference =@ app
end function
```

In this function the Data menu events are all directed at standard functions from the `appframework.sml` library. The `exitviamenu()` function simply calls the `exit()` function, and the `helpabout()` function merely displays a `wxmessagedialog()` call. For full details look at the source code.

Now let's have a look at the tool bar creation code. Like with the menu bar code, the references are added afterwards in the `inittoolbar()` function, but unlike the menu bar, the functions are assigned during the creation of the tool bar.

Example 6.7. The Code for the Tool Bar

```
function buildiconbar(syscolors systemcolors)
  wxbitmap bmp, disbmp
  integer e
  wxtoolbar tb
  wxform f

  e = 0
  tb =@ wxtoolbar.new(16, 16, error=e)

  if tb !@= .nul
    f =@ combos(systemcolors)
    if f !@= .nul
```

```
        tb.insertform(f, name="fileindexcombos")
    end if

    bmp =@ wxbitmap.new("16x16_selfirst.png", "png")
    disbmp =@ wxbitmap.new("16x16_selfirst_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select first \
              record", name="tSelFirst")
    tb!tSelFirst.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_selrwnd.png", "png")
    disbmp =@ wxbitmap.new("16x16_selrwnd_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select \
              rewind", name="tSelRwnd")
    tb!tSelRwnd.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_selprev.png", "png")
    disbmp =@ wxbitmap.new("16x16_selprev_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select \
              previous record", name="tSelPrev")
    tb!tSelPrev.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_selcur.png", "png")
    disbmp =@ wxbitmap.new("16x16_selcur_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select \
              current record", name="tSelCurr")
    tb!tSelCurr.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_selnext.png", "png")
    disbmp =@ wxbitmap.new("16x16_selnext_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select next \
              record", name="tSelNext")
    tb!tSelNext.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_selffwrd.png", "png")
    disbmp =@ wxbitmap.new("16x16_selffwrd_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select fast \
              forward", name="tSelFfwd")
    tb!tSelFfwd.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_sellast.png", "png")
    disbmp =@ wxbitmap.new("16x16_sellast_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select last \
              record", name="tSelLast")
    tb!tSelLast.onclick.function =@ selrec

    bmp =@ wxbitmap.new("16x16_selkey.png", "png")
    disbmp =@ wxbitmap.new("16x16_selkey_disabled.png", "png")
    tb.insert(bmp, disbmp, enabled=.false, tooltip="Select a \
              record by value", name="tSelKey")
    tb!tSelKey.onclick.function =@ selrec

    // Enable these if the form has multiple pages; the
    // changepage() function is already provided

//    bmp =@ wxbitmap.new("16x16_pageprev.png", "png")
//    disbmp =@ wxbitmap.new("16x16_pageprev_disabled.png", "png")
//    tb.insert(bmp, disbmp, enabled=.false, tooltip="Show \
//              previous page", name="tPagePrev")
//    tb!tPagePrev.onclick.function =@ changepage
//
//    bmp =@ wxbitmap.new("16x16_pagenext.png", "png")
//    disbmp =@ wxbitmap.new("16x16_pagenext_disabled.png", "png")
//    tb.insert(bmp, disbmp, enabled=.false, tooltip="Show next \
//              page", name="tPageNext")
//    tb!tPageNext.onclick.function =@ changepage
  end if
end function tb
```

As with the menu bar creation code, the tool bar code is also not particularly complex. The basic approach is to create the empty tool bar, and then add the tools into the tool bar in the order they should appear. Since the very first things that are shown are the table and index combo boxes, and since these are not native tools for the tool bar, they are added by creating a form to host them and then the form is inserted into the tool bar. Following on from there, the tools used for selecting records are all added to the tool bar. Each tool uses two images, one showing what it looks like when it is enabled, and another for when it is disabled. All of the selection functions use the same event handling function, called `selrec()`. The final two items are disabled here, since there is only one page in the form in this example.

The code that creates the combos is very straightforward. It uses a function to create the form and return it to the caller.

Example 6.8. The Code for the Tool Bar Combo Boxes

```
function combos(syscolors systemcolors)
  wxform f
  wxfont font1
  type(wxformcontrol) fc
  sysrgb btnface, comboback, combotext
  integer e

  e = 0
  font1 =@ wxfont.new("MS Sans Serif", 9, error=e)
  btnface =@ systemcolors.colors[COLOR_BTNFACE]
  comboback =@ systemcolors.colors[COLOR_WINDOW]
  combotext =@ systemcolors.colors[COLOR_WINDOWTEXT]

  f =@ wxform.new(width=311, height=24)
  f.setbackgroundrgb(btnface.value)
  fc =@ f.addcontrol(wxformcombo, 1, 1, 150, 19, \
                     edittype="droplist", name="cbFiles")
  fc.onselectionchange.function =@ toolbarcomboevents
  fc.setbackgroundrgb(comboback.value)
  fc.settextrgb(combotext.value)
  fc.setfont(font1)
  fc.setenabled(.false)
  fc.settooltip("Select the table to view")
  fc =@ f.addcontrol(wxformcombo, 156, 1, 150, 19, \
                     edittype="droplist", name="cbIndexes")
  fc.onselectionchange.function =@ toolbarcomboevents
  fc.setbackgroundrgb(comboback.value)
  fc.settextrgb(combotext.value)
  fc.setfont(font1)
  fc.setenabled(.false)
  fc.settooltip("Select the current index for the current table, \
                 or none for sequential access")
end function f
```

As can be seen here, the form controls and the form use the system colors to ensure that they blend in with the system colors as much as possible. They also set tool tip values, like the tools in the tool bar code earlier.

The last piece of this initialization code is the function that initializes the tool bar. It is quite similar to that used to initialize the menu bar, except for the fact that it passes in the appwindow object instead of the application object as a reference. This is primarily because in the case of the tool bar the events more often need fast access to the components of the appwindow object, whereas in the more complex menu routines the application object can be more useful.

Example 6.9. The Code for the Tool Bar Initialization

```
function inittoolbar(wxtoolbar tb, appwindow appw)
  tb!tSelFirst.onclick.reference =@ appw
  tb!tSelRwnd.onclick.reference =@ appw
  tb!tSelPrev.onclick.reference =@ appw
  tb!tSelCurr.onclick.reference =@ appw
  tb!tSelNext.onclick.reference =@ appw
  tb!tSelFfwd.onclick.reference =@ appw
  tb!tSelLast.onclick.reference =@ appw
  tb!tSelKey.onclick.reference =@ appw

// Only uncomment these if the objects have also been created above
//  tb!tPagePrev.onclick.reference =@ appw
//  tb!tPageNext.onclick.reference =@ appw

  tb!fileindexcombos!cbFiles.onselectionchange.reference =@ appw
  tb!fileindexcombos!cbIndexes.onselectionchange.reference =@ appw
end function
```

Since in the code that creates the tool bar the functions were already assigned, in this function there are only a set of statements assigning the appwindow object as the reference for each event handler. As was the case with the definition of the tool bar earlier, there are some lines commented out for working with changing pages. These also need to be uncommented if the form has multiple pages.

Preparing the Form

One of the last things to be done, after initializing the program and opening the form, is to prepare the form for one very important task. There is still a need when creating records to create the unique key, and this will be done in the onnewrecord event of the dataform1 object. This section has two main parts, the code that prepares the form, and the code that creates the unique key value.

Example 6.10. The `prepaddressbookform()` Function

```
function prepaddressbookform(appwindow appw)
  dataform1 form

  form =@ appw.form
  form.onnewrecord.function =@ ab_onnewrecord
  form.onnewrecord.reference =@ appw
```

```
end function
```

This function is very short. It is used only to assign the function and reference to the event. The only reason for separating it into a function is that later there may be other event handlers, as the application grows in complexity, and this way there is already a place for them without overcrowding the `main()` function. The more important part is the function that handles this event. Let's look at it now.

Example 6.11. The `ab_onnewrecord()` Function

```
function ab_onnewrecord(dataform1 me, appwindow appw)
  type(db1record) r
  integer i, e
  sbme1table address

  e = 0
  address =@ appw.app.address
  r =@ address!AddressID.index.select(lastrecord=.true, error=e)
  if r =@= .nul
    i = address.recordcount()
    i = i + 1
  else
    i = r!AddressID + 1
  end if
  me.masterrecord.record!AddressID = i
  me.refresh()
  me!tbFirstnames.setfocus()
end function
```

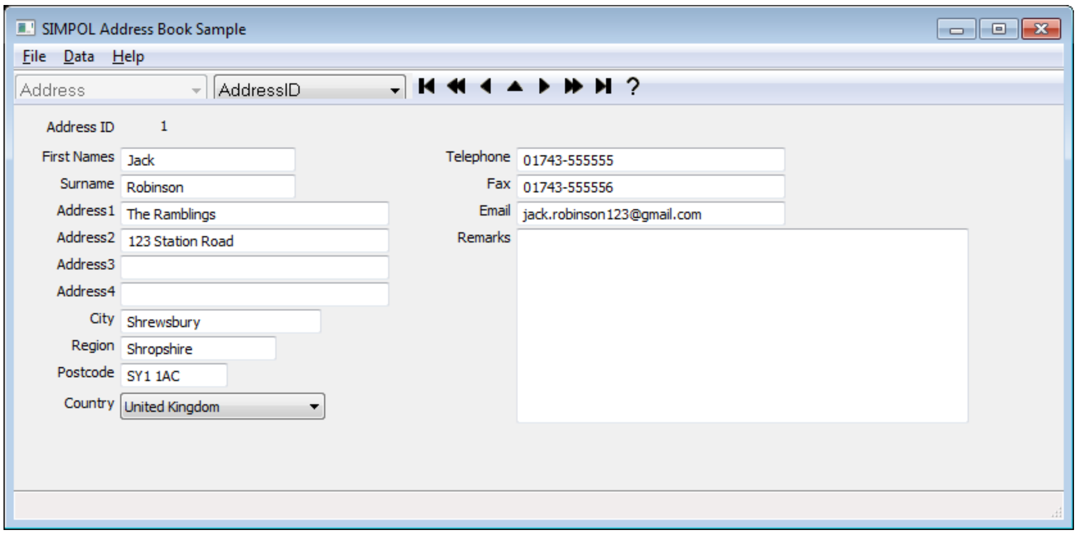
This function is not particularly clever, and it shouldn't be used for a networked application, but in single-user programs it will work just fine. What it does is fairly obvious, it retrieves the last record according to the `AddressID` field's index, and increments that value by one. It then refreshes the form and sets focus to the control that is at the start of the tab order.

Tip

Creating a really powerful function for generating almost perfectly sequential numbers is a fairly non-trivial exercise. Especially if the user can discard the record after creating it. Most approaches use a database table to hold the serial numbers. Typically one record for each table. This allows the standard locking mechanisms to be used to prevent multiple users getting the same number. One approach is to only retrieve the value at the end, while saving, but this can be problematic, especially if there are dependent records that need to have a matching key value inserted. Another approach requires two tables, one for the serial numbers and one for the numbers that have been discarded. It also requires code to handle the discard of a record, so that the number can be placed in the discards table. The discards are then always used first in preference to the main serial number table. In a busy system, there still might be holes in the end of the sequence at any given point, however.

The Finished Product

After all of that work, when we finally run it it shows up looking like the image below:



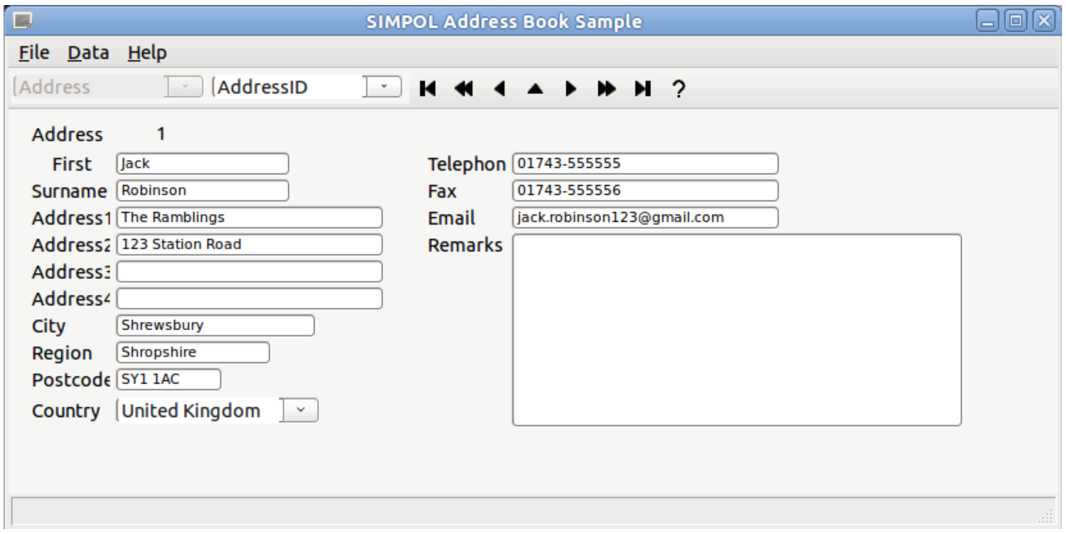
Address ID 1
First Names Jack
Surname Robinson
Address1 The Ramblings
Address2 123 Station Road
Address3
Address4
City Shrewsbury
Region Shropshire
Postcode SY1 1AC
Country United Kingdom
Telephone 01743-555555
Fax 01743-555556
Email jack.robinson123@gmail.com
Remarks

The finished Address Book application

As we can see, everything is now in place. Making changes to the application would be as simple as modifying the form or adding more forms and loading them based on button selections or menu or tool bar items.

A Word About Linux

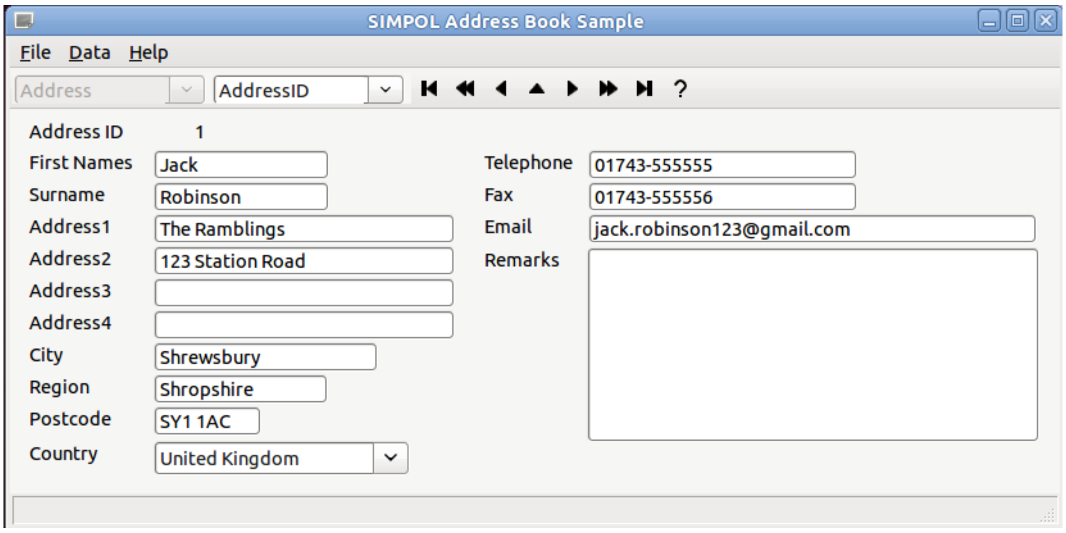
When moving an application to Linux, it is important to recognize that different fonts will be available. Even more important though, is that some Linux window managers will change the fonts to the current theme setting completely replacing the fonts that are used in the form. Also, they may remove or not support attributes like right alignment. As a result, if you intend to produce a version that runs on both platforms, you should either plan your fonts and design accordingly, or use two different forms, one for each platform and design each using the fonts and font sizes for that platform. For example, on Windows the form was designed with 8 point fonts. The font used was actually a placeholder name: "MS Shell Dlg 2", which will become different fonts on different version of Windows. On Ubuntu Linux, the default font was Ubuntu and the size was 11 points. The size, more than the font choice will impact the design, as seen below:



Address 1
First Jack
Surname Robinson
Address1 The Ramblings
Address2 123 Station Road
Address3
Address4
City Shrewsbury
Region Shropshire
Postcode SY1 1AC
Country United Kingdom
Telephon 01743-555555
Fax 01743-555556
Email jack.robinson123@gmail.com
Remarks

The finished Address Book application on Ubuntu Linux

The Ubuntu fonts can be installed on Windows as well, which makes the task of designing the form using the Form Designer much easier. After some basic adjustments, including modifying the code that creates the combo boxes in the tool bar to increase the font size if the OS is not Windows, the new result on Ubuntu Linux can be seen below:



Address ID	1	Telephone	01743-555555
First Names	Jack	Fax	01743-555556
Surname	Robinson	Email	jack.robinson123@gmail.com
Address1	The Ramblings	Remarks	
Address2	123 Station Road		
Address3			
Address4			
City	Shrewsbury		
Region	Shropshire		
Postcode	SY1 1AC		
Country	United Kingdom		

The updated Address Book application on Ubuntu Linux

Summary

In this chapter we have built a database container, and a table to hold our data. We then built a form to allow us to perform easy data-entry. Finally, we wrapped the whole thing in some program code that creates the window, the menu and tool bars, and which can run as a standalone program and act as the basis for a much larger and more complex system. Although we may choose to do some things slightly differently in a larger or in a networked program, we now have a sound foundation on which to build.

More importantly though, by using the supplied sample application together with the `appframework.sml` library, it is possible to quickly build a working prototype application with data-aware forms. The only areas that *must* be modified are:

- The name of the form being opened in function `main()`
- The name of the data source and table being opened in method `addressbookapplication.new()`
- The content of the function `ab_onnewrecord()` (if required – otherwise remove the call to the `prepaddressbookform()` from function `main()`)

The following items *should* be changed as well for a long-term project:

- The name of the addressbookapplication type should correctly reflect the application being built.
- The `prepaddressbookform()` and `ab_onnewrecord()` functions should be replaced with appropriately named functions.
- The `helpabout()` function should show the correct information, and could be replaced with a modal dialog.

Hopefully this chapter and the chapters up until now will have provided you with the tools that you need to make a fast start in the world of SIMPOL programming. Like in a good book, there is something to appreciate right at the start, but the more you investigate, the more there is to discover, if you wish to go there. Post your investigations and questions in the online forum and as a community we can go there together.

Advanced Topics

Now that you have gotten a basic single form and database table package running in single user mode, it is worth thinking about where to go from here. There are a couple of steps that come next:

- Loading other forms
- Changing to a multi-user system

Loading other forms turns out to be fairly easy. The basic call is `appw.openformdirect("myform.sxf")`. The framework takes care of the rest. It is a good idea to open all the required database tables during initialization of the application and ensure they are part of the appwindow object.

Changing to a multi-user system is a bit more complicated. First the database tables need to be opened and shared using a PPCS server. A sample server is included with Superbase NG. It is located in the `SIMPOL\Utilities\simpolserver` directory. Also in that directory is a file called `readme.txt`. That file discusses everything necessary to share the database files via PPCS. Once the files are shared, the remaining change that is required is to open them using a PPCS data source.

```
if bUSEPPCS
  me.ppcs =@ ppcstype1.new(udpport=.nul, error=e)
  if me.ppcs =@= .nul
    wxmessagedialog(appw.w, "Error starting PPCS", sAPPMSGTITLE, \
          "ok", "error")
  else
    src =@ me.opendatasource("ppcstype1", "127.0.0.1:4000", appw, \
          error=e)
    if src =@= .nul
      wxmessagedialog(appw.w, "Error opening the PPCS server", \
          sAPPMSGTITLE, "ok", "error")
    end if
  end if
else
  src =@ me.opendatasource("sbme1", "address.sbm", appw, error=e)
  if src =@= .nul
    wxmessagedialog(appw.w, "Error opening the address.sbm file", \
          sAPPMSGTITLE, "ok", "error")
  end if
end if

if src !@= .nul
  t =@ appw.opendatatable(src, "Address", error=e)
  if t =@= .nul
    wxmessagedialog(appw.w, "Error opening the 'Address' table", \
          sAPPMSGTITLE, "ok", "error")
  else
    me.address =@ t
    ok = .true
  end if
end if
```

The above program listing assumes that a boolean constant called `bUSEPPCS` has been defined earlier in the program. That is all that is required to switch the program to run as a multiple user system, other than a fully licensed database engine, though the 3-user license that is provided with Superbase NG for testing should be sufficient while doing development.

Chapter 7. SIMPOL Business

Introduction

All of the chapters up until now have been leading to a more comprehensive, complex, but also more realistic example of the kind of program people need to build in the average organization. This chapter will have much less text and code, but for that it has a very well documented example program that demonstrates many of the features required of a modern database-based application program.

Tip

Before reading and getting heavily involved in this chapter, it is a good idea to at least read through Chapter 5, *Dialog-Style Programs* and Chapter 6, *GUI-Style Database Programs*. Many of the techniques for working with the Form Designer and for creating database-oriented programs are covered in those chapters.

In this chapter we will discuss the features and special techniques used in the SIMPOL Business example program. This example consists of several database tables, four forms, and both a Quick Report and a Graphic Report. The basic design is a typical order entry system with the usual four tables plus a couple of extras. Here is a list of the database tables that are included:

- COUNTRY
- CUSTOMER
- ORDERDTL
- ORDERMST
- PRODUCT
- SERNO

The main tables are the CUSTOMER, ORDERDTL, ORDRMST, and PRODUCT. The COUNTRY table is a very carefully designed table that contains all of the current world country names, ISO-3166 2-character code, internet domain code, CEPT Code, capital city, currency code (3-letter), and the vehicle license plate international ID code. Only the country code is stored in the records from the CUSTOMER and ORDERMST tables. The SERNO table contains a record for each of the other tables, with the table name as the unique key and the current serial number value as the only other field.

There are also the four forms, one each for the CUSTOMER, PRODUCT, and ORDERMST tables with a detail block on the ORDERMST form containing the order lines from the ORDERDTL table. The fourth form is used for creating and editing entries in the ORDERDTL table and is called from buttons on the ORDERMST form.

Special Features

In comparison with the simpler Address Book example from the previous chapter, the SIMPOL Business application adds a number of new capabilities:

- Switching forms and selecting a related record in the target table
- Using preventfocusmode in an application to control user access to the data
- Using the onsave event to do calculations when saving a record and to hide some buttons and enable others on the form
- Using the onchangerecord event to detect entering data-entry mode to disable and show buttons on the form

- Using the ondelete event to ensure that all the related detail records in ORDERDTL can be deleted before allowing the record in ORDERMST to be deleted
- Adding, editing, and deleting records in a detail block
- Using a Graphic Report for an invoice
- Using a Quick Report as a sales report
- Using a labels definition to output customer mailing labels, includig using a call to the new `choicelistdialog()` for the output destination
- Integrating record view, table view, form view, field selection lists, the filter GUI, and the Quick Report GUI
- Getting the user to select a record using the new `drilldown()` function
- Using a serial number table to retrieve a unique serial number for reliability even in a network environment
- Showing one value in a combo box list but assigning an alternate
- Using the correct public data directory on modern Windows operating systems

All of these features are well-documented in the source code of the SIMPOL Business application. However we will still discuss a few of the more interesting features in the following sections.

Working With the dataform1detailblock

About the Design of Detail Blocks

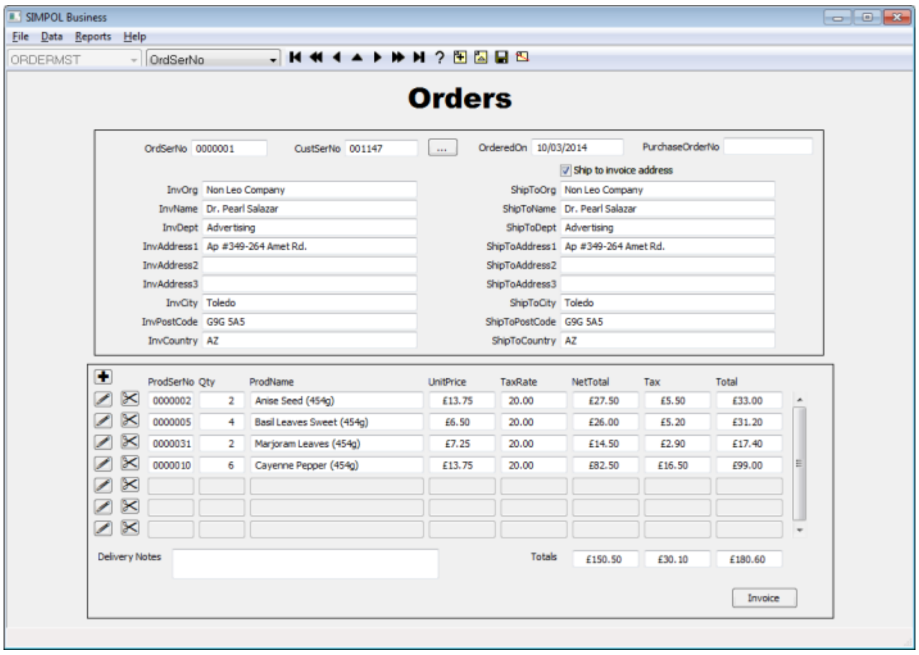
In our initial design and implementation of detail blocks in SIMPOL we recognized that in the original version in Superbase some things had not gone well. Although it was possible to add records as long as there were less records than visible rows, once the visible rows were filled adding records became difficult. There was also no support for deleting records from the detail block. The ability to nest them up to 8 levels deep was troublesome when trying to use them for reliable data-entry.

As a result many Superbase users have been forced to come up with their own solutions to these problems over the years. The solutions usually were either to use a dialog for adding and editing data, or to add a special set of controls on the form where data was created or modified. Most solutions also added a set of buttons to the left of the rows to edit or delete the row data.

During our initial design and implementation for SIMPOL, we decided to make the detail block read-only to avoid the need to wrestle with these issues since, to start with, we just wanted to get a reliably working read-only implementation. There are numerous issues to resolve with something like this if it is allowed to be read/write. We decided to add the ability to modify the content of the detail block under program control. The early versions had some limited ability to do this, but with the 1.8 release, we decided to commit to a detail block that could be completely managed under program control. We added the necessary methods to add, edit, and delete entries.

Adding New Records to Detail Blocks

For this project we chose to use the dialog method for data-entry. New records are created using the same dialog window and form (which is a normal dataform1 form). The only difference is the record is a new one rather than an existing one. In this design, when the order record is created, we disable the buttons that allow the creation, editing, or deletion of detail block records. After it has been saved, these buttons are again enabled. This ensures that the detail block records are created based on an existing order record. Otherwise we would have had the problem of ensuring the records are not saved until the order itself is saved. Below is an image of the orders form.



OrdSerNo	0000001	CustSerNo	001147
OrderedOn	10/03/2014	PurchaseOrderNo	

☑ Ship to invoice address

InvOrg	Non Leo Company	ShipToOrg	Non Leo Company
InvName	Dr. Pearl Salazar	ShipToName	Dr. Pearl Salazar
InvDept	Advertising	ShipToDept	Advertising
InvAddress1	Ap #349-264 Amet Rd.	ShipToAddress1	Ap #349-264 Amet Rd.
InvAddress2		ShipToAddress2	
InvAddress3		ShipToAddress3	
InvCity	Toledo	ShipToCity	Toledo
InvPostCode	G9G 5A5	ShipToPostCode	G9G 5A5
InvCountry	AZ	ShipToCountry	AZ

ProdSerNo	Qty	ProdName	UnitPrice	TaxRate	NetTotal	Tax	Total
0000002	2	Anise Seed (454g)	£13.75	20.00	£27.50	£5.50	£33.00
0000005	4	Basil Leaves Sweet (454g)	£6.50	20.00	£26.00	£5.20	£31.20
0000031	2	Marjoram Leaves (454g)	£7.25	20.00	£14.50	£2.90	£17.40
0000010	6	Cayenne Pepper (454g)	£13.75	20.00	£82.50	£16.50	£99.00
				Totals	£150.50	£30.10	£180.60

Delivery Notes

The Orders Form from SIMPOL Business.

The key bit of code for both adding and editing records in the detail block can be found in two of the included functions: `addorderline()` and `editorderline()`. They both call the function `doeditaddorderline()` to actually present the dialog box and handle the interaction with the user. At the end that function returns a type(db1record) object and sets the boolean `saved` variable to `.true` if the user saved the record. Below is the `addorderline()` function.

Example 7.1. The `addorderline()` function of the SIMPOL Business program

```
function addorderline(dataform1button me, appwindow appw)
  boolean saved
  type(db1record) r
  dataform1record rec
  dataform1table table
  integer ordserno, e
  dataform1detailblock dtb
  dataform1recordset rset

  saved = .false
  e = 0
  ordserno = me.form.masterrecord.record!OrdSerNo
  r =@ doeditaddorderline(appw, .true, saved=saved, \
                        ordserno=ordserno)
  if saved
    // Here we need to add the row to the detail block and move
    // the current row pointer
    dtb =@ me.form!dtbOrderLines
    if dtb !@= .nul
      rset =@ dataform1recordset.new()
      table =@ me.form.findtable(r.table.tablename)
      rec =@ dataform1record.new(r, table, error=e)
      // Here we are placing the record in the record set as the
      // master record
      rset.records[1] =@ rec
      dtb.addrowdata(rset, error=e)
      calculateordertotals(me.form, dtb, appw)
    end if
  end if
end function
```

The important point of this is the place in the code where the record is added to the detail block.

Note

A detail block row is represented by a dataform1recordset object. This contains a records property of type array. Each element in the records array is of type dataform1record. If the detail block contains multiple linked records that are linked 1:1 (for example a product name that is not stored in the detail record but which is only looked up via the product ID), then for each linked table there will be an additional dataform1record object in the records array.

In the example above, we are working with a simple detail block consisting of only the detail table record in each row. To write the new record to the detail block, we create a new record set, create a new dataform1record object using the record that was returned, and then assign the dataform1record object to the first element of the record set's records array. Once our preparation is complete, we call the `addrowdata()` method of the dataform1detailblock object. Also, since we are managing the totals of several of the columns in the `ORDERMST` record, we also call the `calculateordertotals()` function that calculates the totals to update the values in that record and to show them on the form.

Editing Records in a Detail Block

The code that handles the editing of the detail block is very similar to that which adds a new record:

Example 7.2. The `addorderline()` function of the SIMPOL Business program

```
function editorderline(dataform1button me, appwindow appw)
  integer row, e, orditemno
  dataform1detailblock dtb
  dataform1recordset rset
  dataform1record rec
  type(db1record) r
  boolean saved
  sbapplication app

  saved = .false
  app =@ appw.app
  e = 0
  dtb =@ me.form!dtbOrderLines
  if dtb !@= .nul
    row = .toval(me.name, nondigits(me.name), 10)
    if row >= 1 and row <= dtb.rows
      rset =@ dtb.getrowdata(row, error=e)
      if rset =@= .nul
        wxmessagedialog(appw.w, "Error no row data available", \
                        sAPPMSGTITLE, "ok", "error")
      else
        rec =@ rset.records[1]
        if rec =@= .nul or rec.record =@= .nul
```

```
          wxmessagedialog(appw.w, "Error record not found in the \
                          row data", sAPPMSGTITLE, "ok", "error")
        else
          r =@ rec.record
          orditemno = r!OrdItemNo
          r =@ .nul
          r =@ doeditaddorderline(appw, .false, orditemno, saved)
          if saved
            // Update the specific row in the detail block
            rec.record =@ r
            dtb.setrowdata(row, rset, error=e)
            calculateordertotals(me.form, dtb, appw)
          end if
        end if
      end if
    end if
  end if
end function
```

In the preceding example the name of the button control contains the row number and that is retrieved by using the `.toval()` function and by declaring all of the non-digit content using the `nondigits()` function. Then we call the `getrowdata()` method of the detail block passing in the row number to retrieve the record set representing that row. We access the dataform1record that contains the detail block record from the record set and use that to read our unique record ID, the value of the `OrdItemNo` field.

We then clear the record variable by setting it to `.nul` and call the `doeditaddorderline()` function passing in the `orditemno` variable. If the user has saved the changes to the record, then we need to replace the old version of the record in our dataform1record object with the updated version. Then all that is left is to call the `setrowdata()` method of the detail block and as with the new record, we need to call the `calculateordertotals()` function to update the totals in the master record and display them on the screen.

Deleting Records in a Detail Block

All that remains with our detail block is to be able to delete records from it. The program code that does that is very similar to that used for editing:

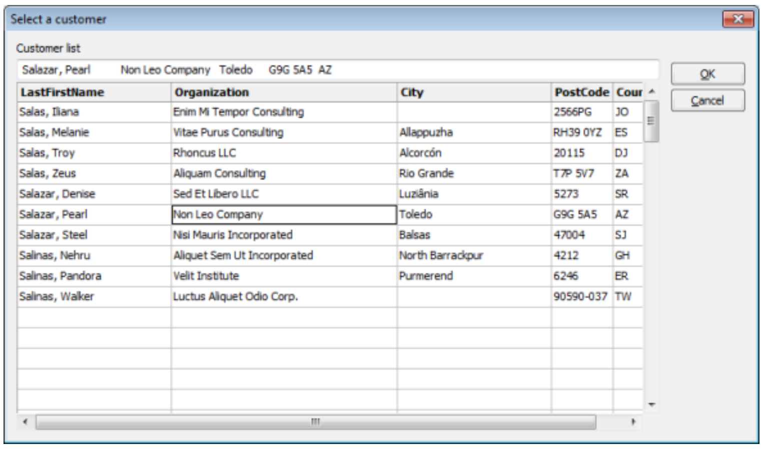
Example 7.3. The `addorderline()` function of the SIMPOL Business program

```
function deleteorderline(dataform1button me, appwindow appw)
  integer row, e
  dataform1detailblock dtb
  dataform1recordset rset
  dataform1record rec

  e = 0
  dtb =@ me.form!dtbOrderLines
  if dtb !@= .nul
    row = .toval(me.name, nondigits(me.name), 10)
    if row >= 1 and row <= dtb.rows
      rset =@ dtb.getrowdata(row, error=e)
      if rset =@= .nul
        wxmessagedialog(appw.w, "Error no row data available", \
                        sAPPMSGTITLE, "ok", "error")
      else
        rec =@ rset.records[1]
        if rec =@= .nul or rec.record =@= .nul
          // wxmessagedialog(appw.w, "Error record not found in \
          //                  the row data", sAPPMSGTITLE, "ok", \
          //                  "error")
        else
          rec.lock(error=e)
          if e != 0
            wxmessagedialog(appw.w, "Error locking the record", \
                            sAPPMSGTITLE, "ok", "error")
          else
            rec.delete(error=e)
            if e != 0
              wxmessagedialog(appw.w, "Error deleting the record",\
                              sAPPMSGTITLE, "ok", "error")
            else
              dtb.removerowdata(row, error=e)
              calculateordertotals(me.form, dtb, appw)
            end if
          end if
        end if
      end if
    end if
  end if
end function
```

Just as was done in the edit code, first we transform the control name into the row number and then we use that to retrieve the record set representing that row. After extracting the dataform1record object from the record set, we lock it and assuming that succeeded, we call the `delete()` method of the dataform1record object. If that also succeeds (it should), the `removerowdata()` method of the detail block object is called to remove the actual record set for that row and to adjust the scroll position of the visible rows on the form.

Using the `drilldown()` Function

To make it easy to select the correct customer for an order, the new `drilldown()` function was brought into the project. It requires a bit of set up, but provides a fast and efficient method of finding a target record. An example of it can be seen below.



Salazar, Pearl   Non Leo Company Toledo   G9G 5A5 AZ

LastFirstName	Organization	City	PostCode	Cour
Salas, Ilana	Enim Mi Tempor Consulting		2566PG	JO
Salas, Melanie	Vitae Purus Consulting	Allappuzha	RH39 0YZ	ES
Salas, Troy	Rhoncus LLC	Alcorcón	20115	DJ
Salas, Zeus	Aliquam Consulting	Rio Grande	T7P 5V7	ZA
Salazar, Denise	Sed Et Libero LLC	Luziânia	5273	SR
Salazar, Pearl	Non Leo Company	Toledo	G9G 5A5	AZ
Salazar, Steel	Nisi Mauris Incorporated	Balsas	47004	SJ
Salinas, Nehru	Aliquet Sem Ut Incorporated	North Barrackpur	4212	GH
Salinas, Pandora	Velit Institute	Purmerend	6246	ER
Salinas, Walker	Luctus Aliquet Odio Corp.		90590-037	TW

The drill down window searching for a customer

As the user types in the top edit control, the system detects that and searches against the index passed as the search index. It then fills the grid with data up to the maximum number of desired rows. The code that does this is shown here:

Example 7.4. The `findcustomer()` Function for the Orders Form

```
function findcustomer(dataform1button me, appwindow appw)
  type(db1record) r
  type(db1index) idx
  sbapplication app
  tdisplayformats dispfmt
  array dispflds, colwidths
  integer e

  if not me.form.preventfocus
    app =@ appw.app
    dispfmt =@ app.displayformats
    idx =@ app.tables.customer!LastFirstName.index
    dispflds =@ array.new()
    dispflds[1] =@ app.tables.customer!LastFirstName
    dispflds[2] =@ app.tables.customer!Organization
    dispflds[3] =@ app.tables.customer!City
    dispflds[4] =@ app.tables.customer!PostCode
    dispflds[5] =@ app.tables.customer!CountryCode

    colwidths =@ array.new()
    colwidths[1] = 150
    colwidths[2] = 220
    colwidths[3] = 150
    colwidths[4] = 60
    colwidths[5] = 50

    e = 0
    r =@ drilldown(appw.w, 730, 400, idx, 100, 1, "Select a \
               customer", "Customer list", \
               dispflds=dispflds, colwidths=colwidths, \
               defboolean=dispfmt.defboolean, \
               definteger=dispfmt.definteger, \
               defnumber=dispfmt.defnumber, \
               defdate=dispfmt.defdate, \
               deftime=dispfmt.deftime, \
               defdatetime=dispfmt.defdatetime, \
               datelocale=app.SBLlocale.datelocale, \
               numlocale=app.SBLlocale.numlocale, error=e)
    if r !@= .nul
      updatecustonorderform(me.form, r, appw)
    end if
  end if
end function
```

The code should be fairly obvious, we first check to make sure we are in data-entry mode by testing the me.form.preventfocus value. If it is equal to `.true` then we are not in data-entry, so ignore clicks.

Note

In preventfocusmode in the ongotfocus event of the dataform1 controls if the preventfocus property is equal to `.true` then nothing happens and focus is ignored. This does not prevent the onclick event of buttons from firing, however, so if they should not fire at all times it is necessary to test for the state of this property.

To reduce the amount of typing we declared the `dispfmt` variable and assigned the app.displayformats property to it. We then acquire a reference to the index object we wish to use for searching, produce an array to hold the field object references for the fields we wish to display in the grid, and assign the column widths to the `colwidths` array. The `colwidths` array is optional. If they are not passed the column widths will be derived from the table information. Finally the call is made to the `drilldown()` function and if the user clicks on OK, then it will return the selected record object, otherwise it will return `.nul`.

Storing Data Correctly in Modern Windows Systems

As of Windows Vista it became difficult to modify data stored in the `Program Files` directory. Although the system doesn't cause an error when a program writes data there, what actually happens is the data is not written to that location, but instead it is written to a location below the user directory. The location is typically something like: `\User\AppData\Local\VirtualStore\Program Files\`.... Although that may not matter in a single-user system (though you may not be backing up the data correctly), in a multi-user system, where more than one person logs onto the same PC, that would mean that each user would have their own copy of the data, and changes from one would not appear in the data of the other.

The solution to this mess, is to store the data in a publicly accessible location. On Vista and later, that is the `\Users\Public\Documents` directory. There is a function in the application framework called `getpublicdatadir()` that can be used for this purpose. A small part of the initialization code from the `sbapplication.new()` method demonstrates how this is used in the SIMPOL Business application.

Example 7.5. The `findcustomer()` Function for the Orders Form

```
e = 0
datadir = getpublicdatadir(error=e)
if datadir <= ""
  wxmessagedialog(appw.w, "Error retrieving data directory", \
                  sAPPMSGTITLE, "ok", "error")
else
  dirsep = getdirectorysepchar()
  me.dirsep = dirsep
  me.startdir = trailingdirsep(getcurrentdirectory())
  datadir = trailingdirsep(datadir) + sAPPNAME + dirsep
```

The preceding fragment of code shows the approach. The `datadir` is constructed by combining the return value from `getpublicdatadir()` with the application name followed by the directory separator character. On most systems this will be: `C:\Users\Public\Documents\SIMPOL Business\`. The installer will have created this directory and copied the database files plus the `SIMPOL Business.ini` file into it.

Summary

In this chapter we have discussed some of the more interesting techniques used in the SIMPOL Business sample application. There are more to be discovered, and the easiest way to do that is by opening the project in the IDE and looking at the source code. Try running it in debug mode and see how various things work. There is no substitute for getting into the code. It is the best method to learn about how things work. An interesting thing to do would be to change the parameters to use PPCS and set up the database with the `simpolserver.exe` and an appropriate configuration file.

Chapter 8. SIMPOL Server

About the SIMPOL PPCS Server Programs

In this chapter we will discuss the multi-user database server programs currently shipped with SIMPOL, `simpolserver.exe`/`simpolserver.smp` and `guisimpolserver.exe`. The first is designed to run without a user-interface, from a command line prompt or as a service (more later). The second is designed to run on a logged in server as a window program with buttons for sharing the tables, stopping sharing, reorganizing the tables, backing them up and restoring from the back up. Both take a single parameter on the command line to tell them which configuration file to load. Both do the job fine. Using these or variations of these we have been deploying systems for years on both Windows and Linux. In the next sections we will go into each of them in more detail. Before we do that though, since both make use of the same configuration file format, let's examine that.

The Configuration File

The configuration file used by both servers is based on the ini file format. There are two sections in this file. Below is an example of the file format. The file format has changed since earlier versions, but the old format is still supported. Just don't include a `cfgversion` parameter and don't make use of any of the new parameters.

Example 8.1. A Sample SIMPOL Server Configuration File

```
[Server]
cfgversion=2
port1=4000
txfactor1=0
#port2=4001
#txfactor2=6
tcpport=4002
logfilename=samplelog.txt
bz2libdll=c:\simpol\bin\bzip2.dll
archiveroot=SIMPOL Data Backup
backupdir=C:\Users\Public\Documents\SIMPOL\backup
title=SIMPOL Server
deflocktimeout=10000000

[Files]
1=c:\simpol\utilities\simpolserver\adrb.sbm,locktimeout=120000000
```

The `[Server]` Section of the Config File

In the `Server` section, the `port` entry can be repeated multiple times, but each must end in a different sequential value starting with `1`. The server will listen on each of the ports listed. The `txfactor` value is used to reduce the transmission speed of the server. This is important especially when working with Superbase clients, where the client cannot process the data fast enough to keep up, resulting in lost packets. Also, when debugging it may be necessary to slow down for SIMPOL to `6` or even `9` or `10`. The `tcpport` is used so that the serverclose.smp (or .exe) can request the simpolserver to shutdown.

As can be seen above, the entries for `port2` and `txfactor2` are commented out. You don't need `port2` and `txfactor2` (but you always need both together), unless you are coping with systems with differing network access speeds, like a LAN and a WAN. You might set the LAN txfactor to 0 and the WAN txfactor to 6, for example. Tables with larger numbers of fields can take longer per record, so you may need to increase the txfactor to support the time it takes for the record or file definition to be processed.

The `tcpport` is used to host a TCP/IP server that can be contacted using the `serverclose.exe` (or .smp) file. As a seurity precaution, it must be run on the same physical machine or it will be ignored. The command it sends is simply a `QUIT` message, but this ensures that the server will correctly shut down and flush all changes to disk.

If the `logfilename` parameter is assigned, then a log will be produced and written to a file of the same name.

When using the `guisimpolserver.exe` or the `simpolserver.exe` with the `simpolserverclient.exe`, the following three entries are required for doing back up and restore of data:

- `bz2libdll`
- `archiveroot`
- `backupdir`

The first provides the name and location of the BZip2 DLL that is used for compressing and decompressing the data files. The `archiveroot` defines the root file name to which the date and time will be appended. The last item identifies the location where the back up files will be stored.

The `deflocktimeout` entry allows the lock time out value to be set to a standard value, which will be inherited by all of the entries in the `[Files]` section, unless expressly overridden.

The `[Files]` Section of the Config File

The format of the `Files` entries is as follows: `<index value>=<path and filename>, locktimeout=.inf, hidden=f, reccount=f, codepage=850, r=, rc=, rl=, rlc=, rlm=, rld=, rlcd=, rlcm=, rlmd=, rlcmd=`. Here they are listed out:

- `<index value>`
- `<path and filename>`
- `locktimeout`
- `hidden`
- `reccount`
- `codepage`
- `r`
- `rc`
- `rl`
- `rlc`
- `rlm`
- `rld`
- `rlcd`

- `rlcm`
- `rlmd`
- `rlcmd`

The `index` value must be in sequential order starting with `1` and not contain duplicates. As soon as a value is missed the program stops reading files.

The path and file name for each `*.sbm` file adds that container file and all of its tables (except for the system tables) to the server.

The `locktimeout` value defaults to `.inf` (never unlocks from the server side), unless it is overridden with a value assigned to the `deflocktimeout` entry in the `[Server]` section. This value should be set to be appropriate for the individual table and its use. The value is in microseconds, so to automatically unlock in 12 seconds, use: `12000000`. It applies to all tables located in the same container file.

The default `hidden` value is `f` (false). To enable it, set it to `t` (true). Tables can be hidden so that someone connecting to the PPCS server cannot list them. If they know the table name, then they can still open the it, unless it is password protected.

The `reccount` default value is `t`. To suppress the determination of the record count for a table, set it to `f` (false). This will assign the maximum record count to the table and not try to calculate it. On very large tables calculating the record count can take some time, making server starts slow.

PPCS operates in the DOS code page normally (for historical reasons having to do with compatibility with the PPCS protocol of its predecessor (Superbase). The default code page for PPCS is 850 (Latin 1). This can be changed to another code page by setting this parameter. This is the list of supported code pages: 437, 720, 737, 775, 850, 852, 855, 857, 862, 866, 874, 1258.

The `r=,rc=,rl=,rlc=,rlm=,rld=,rlcd=,rlcm=,rlmd=,rlcmd=` parameters are all passwords. To password protect the database table purely for access, it is only necessary to apply a read password. More fine-grained control can be had using the other password combinations, if desired. The letters stand for the following capabilities with respect to records:

- `r` – read
- `c` – create
- `l` – lock
- `m` – modify
- `d` – delete

Please note that if you decide to use multiple passwords to access these tables from different users with different access rights, then the program code needs to be able to cope with any errors that may occur.

Working with `simpolserver.exe`

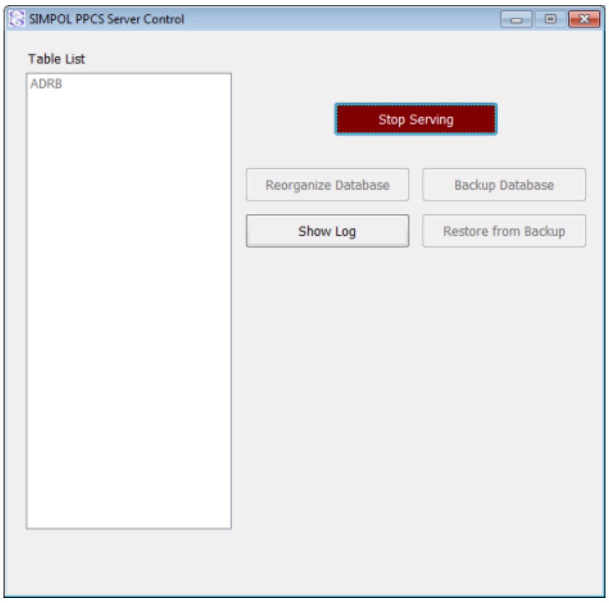
The `simpolserver.exe` program is designed to run without any user interface. It takes a single argument on the command line, which is the name of the configuration file from which it should read the information it needs to run: which port number(s) to listen on, the name of the log file to create, the name(s) of the database files that should be shared and the parameters for each of them. If it is started without any parameters it will search for the file `simpolserverinfo.cfg` in the same directory from where the exe file was started. If it hits an error starting up it will output that information as a return value to the console and if it got far enough there may be some information in the log file.

It can also be run using the `simpolserver.smp` file, in which case it should definitely be run by passing the configuration file name to it as a parameter. This is the program to use when running on Linux.

When running this program, there is a companion program that can provide the user-interface for the server. That program is delivered as `simpolserverclient.exe`. It is currently only available for Windows. It will present a user-interface that looks more or less identical to that of the `guisimpolserver.exe` program, but it acts only as a front-end to the server version. It can also start and stop the sharing of tables, as well as perform a reorganize, back up, restore and can view the log. The `simpolserverclient.exe` takes a single parameter, which is the TCP/IP port where the server is listening. If no parameter is provided, it will search the directory where the exe file was located when it started for a file called `simpolserverinfo.cfg` and if it finds it will read the TCP/IP port value from there.

Working with `guisimpolserver.exe`

The `guisimpolserver.exe` program was designed as a desktop program to be run on a logged-in server. It gives much more fine-grained control than our original `simpolserver.smp` program. With the release of the new version of the `simpolserver.exe` and its ability to be controlled using `simpolserverclient.exe`, there are fewer reasons to choose this version, though it is a much simpler design and may have less issues in a working environment. For one thing, it is all self-contained. On the other hand, if left running for a very long time, it might be less stable than a version that does not have the user-interface components included and running. For now, pick the one that suits you best. Since they both support the same configuration file format, there is no difficulty moving from one version to another. Below is an image showing the user-interface. This image is from the `simpolserverclient.exe` program, but there is no obvious difference between them.



The user interface for `guisimpolserver` and `simpolserverclient`

Running `simpolserver.exe` as a Service

In a production environment, you would ideally want the database server to start when the server starts, and to shut down gracefully when the server shuts down. Using Linux, this is quite easy depending on your distribution. It is fairly trivial to add `/usr/bin/smprun /home/me/simpolserver.smp /home/me/simpolserverinfo.cfg` to the start up of the server and `/usr/bin/smprun /home/me/serverclose.smp 12345` to the shut down of the server (typically something like `local.start` and `local.stop` in `/etc/conf.d/` or `baselayout1.start` and `baselayout1.stop` in `/etc/local.d/`). Linux makes a lot of things easier, which is why we only have one loader program for both console and windowing programs. On Windows, this is all much more complicated.

We created a special program that can run a specified program as a service, with a second program that can be run to stop the service. The program is called `svcrunnr.exe`. To install the service running program, from the command line run this: `svcrunnr.exe -install`. To remove the service, use this command line: `svcrunnr.exe -remove`. Before you start the service, it is useful to check that everything is correctly configured.

Once the service is installed, it is important to check its configuration file. The name of that file is: `svcrunnr.exe.ini`. This is a very simple file and it needs to be located in the same directory as the `svcrunnr.exe` program. It contains two sections, each with only one entry:

Example 8.2. A Sample `svcrunnr.exe` Configuration File

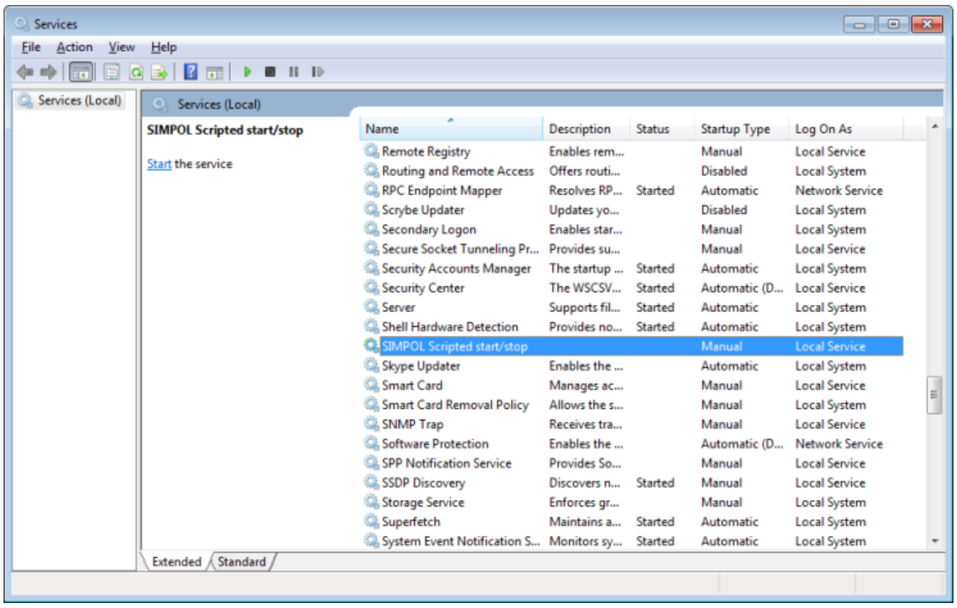
```
[Startup]
Command=C:\SIMPOL\bin\simpolserver.exe

[Shutdown]
Command=C:\SIMPOL\bin\serverclose.exe
```

Normally, there should be no reason to change this configuration file unless the locations are incorrect. It is specific to the installation of the `simpolserver.exe` suite of programs.

Note

Only one instance of this service can be running and/or installed concurrently, so there is no easy use that can be made of the service runner for other purposes. At some point we will release a more powerful service loader for SIMPOL.

After checking the configuration file and after making sure that the `simpolserverinfo.cfg` is correctly set up (because this method will *always* use this file name), if the server should run automatically, then it needs to be set to do so. Upons installation it is set up to run manually only. To change this, in a console window, or in the Start menu enter `services.msc` in the search window and press **Enter**. The services program will start. Navigate in the list to the `SIMPOL Scripted start/stop` entry as shown below:



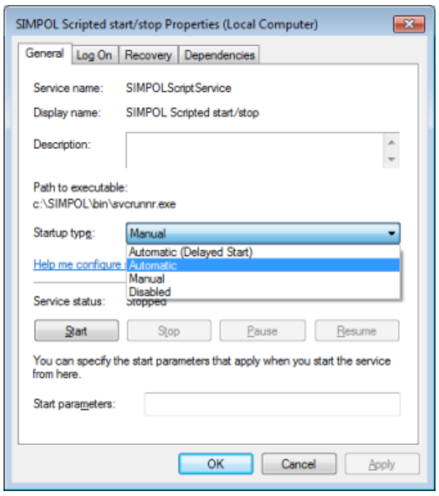
SIMPOL Scripted start/stop

Start the service

Name	Description	Status	Startup Type	Log On As
Remote Registry	Enables rem...		Manual	Local Service
Routing and Remote Access	Offers routi...		Disabled	Local System
RPC Endpoint Mapper	Resolves RP...	Started	Automatic	Network Service
Scrybe Updater	Updates yo...		Disabled	Local System
Secondary Logon	Enables star...		Manual	Local System
Secure Socket Tunneling Pr...	Provides su...		Manual	Local Service
Security Accounts Manager	The startup ...	Started	Automatic	Local System
Security Center	The WSCSV...	Started	Automatic (D...	Local Service
Server	Supports fil...	Started	Automatic	Local System
Shell Hardware Detection	Provides no...	Started	Automatic	Local System
SIMPOL Scripted start/stop			Manual	Local Service
Skype Updater	Enables the ...		Automatic	Local System
Smart Card	Manages ac...		Manual	Local Service
Smart Card Removal Policy	Allows the s...		Manual	Local System
SNMP Trap	Receives tra...		Manual	Local Service
Software Protection	Enables the ...		Automatic (D...	Network Service
SPP Notification Service	Provides So...		Manual	Local Service
SSDP Discovery	Discovers n...	Started	Manual	Local Service
Storage Service	Enforces gr...		Manual	Local System
Superfetch	Maintains a...	Started	Automatic	Local System
System Event Notification S...	Monitors sy...	Started	Automatic	Local System

The Services window showing the svcrunnr entry

Double-click on the entry in the list and in the pop up window change the startup type to Automatic so that it runs when the computer starts, then click on the Start button to start it running. See the image below:



SIMPOL Scripted start/stop Properties (Local Computer)

General | Log On | Recovery | Dependencies

Service name: SIMPOLScriptService

Display name: SIMPOL Scripted start/stop

Description:

Path to executable:
c:\SIMPOL\bin\svcrunnr.exe

Startup type: Manual

Automatic (Delayed Start)
Automatic
Manual
Disabled

Help me configure

Service status: Stopped

You can specify the start parameters that apply when you start the service from here.

Start parameters:

The `SIMPOL Scripted start/stop` window showing the set up

That's it! The database engine will now run automatically when the server starts, and will automatically shutdown when the server is shutdown. To see what is happening and to control it, the `simpolserverclient.exe` program can be run.

SIMPOL Server Summary

In this chapter we have learned about the multi-user database server programs supplied with the SIMPOL product. We have learned how to configure them, start them, stop them, and use them for doing data back up and restore, as well as table maintenance using the reorganize command. Finally, we have learned how to use the `svcrunnr.exe` program to run the `simpolserver.exe` as a service so that it will always be available, even when the computer has just started and has not been logged in.

Chapter 9. Web Server Programs

Introducing World Wide Web Server Programming

SIMPOL's earliest interactive program capabilities were as a web server application. This was done because it was the easiest way to build programs that were able to interact with the user, since we didn't need to worry about building user interface components. As part of the implementation, we decided to support the CGI standard, ISAPI (Microsoft's proprietary interface for IIS) and something called Fast-CGI, which is a high-performance version of CGI that can handle larger loads. It does this by allowing the program code to remain loaded for a certain amount of time, so that subsequent calls to the same program do not need to also go through initialization.

There are typically two approaches to web application programming, page-centered and program-centered. Examples of page-centered programming are typically found in PHP and ASP programs. The page is the focus of the development. This works well for people who are mainly graphics designers and who want to add a little bit of code to their pages. The other approach tends to be much more about the application code, which will, based on the current state, display any of a number of pages. Both are valid methods of working, though application programmers may find the page-centric approach difficult to follow.

SIMPOL supports to some degree both styles. The IDE has support for a file format called `*.smz` which is a form of HTML with embedded SIMPOL code. During the compilation phase these files are converted to program code with embedded HTML. In the standard versions used in ASP and PHP, these pages are processed by the server when they are requested, whereas in SIMPOL they are already compiled to byte code.

The hardest part about web server programming is debugging something like this, which runs in the server, since you can't control the execution. We solved this problem way back in 2003, by building a special program to act as the loader for the CGI program, and which then connects to the SIMPOL IDE to debug the program. This works extremely well and makes debugging the program quite easy. This loader program is called `sbngidecaller.exe`.

Styles of Web Server Application

There are a number of differences between the different web server loading methods. Both the CGI and ISAPI approaches use a `main()` function that takes a cgicall *`cgi`* parameter. The main difference between the two is what the current directory is. For CGI programs, the current directory is the same as the location of the program that is being loaded, while for ISAPI programs, they normally start in the `%SYS32` directory.

The Fast-CGI program is quite different to the other two, though like with CGI, the location of the program file is also the location of the current directory. Instead of a `main()`, there are three separate functions. They are: `fcgiinit()`, which takes no parameters, `fcgi()`, which takes a cgicall *`cgi`* parameter, and an optional type(*) <*`paramname`*> parameter, which is the return value from the `fcgiinit()` function. Finally, there is an `fcgiterm()` function that takes the return value from the `fcgiinit()` function to allow any required clean up to take place. One example of this might be if a dynamically loaded library is opened during initialization and in that some memory is reserved. Then during the clean up phase that memory could be released. In most SIMPOL web server programs, there will be little need for the final function, except as an empty function so that it can be called although it will do nothing.

It is also worth mentioning that with advent of what has been called *Web 2.0* that by using client-side JavaScript and a capability known as AJAX (Asynchronous JavaScript and XML) a new approach to web server applications has arisen, that are more about handling data requests and sending back the data in a specific form. This technique makes use of the `XMLHttpRequest` which, in spite of its name, does not need to use XML. It can transfer data in other ways beside using XML. The interesting part of this approach is that of sending pages back every time a call is made to the web server, the application running in the web browser may only send out data requests to refresh what is shown on the page being displayed. This is far more efficient than sending along the entire page each time you send the data. There are two aspects of Web 2.0 really, one is related to this ability to update the data on the page without resending the page, the other has to do with the use of a modern set of libraries to make web applications look more like desktop applications and is embedded into various JavaScript frameworks. JavaScript is still a frustratingly messy language, but it is showing signs of growth. It remains to be seen how long it will take before it becomes easy to create an application the way *you* want, rather than the way *they* want.

Valuable Reading References Within the SIMPOL Documentation

The *SIMPOL IDE Users Guide* contains a section in Chapter 3, *The SIMPOL Project*, dedicated to what is called “SIMPOL Server Pages”. This fairly extensive section discusses the various aspects of working with the IDE and web server applications.

In the *SIMPOL IDE Quick Start Manual*, in Chapter 3 *Writing Web Server Programs With SIMPOL*, a complete example of writing a web server program is presented, including setting up the Apache web server and debugging the application. This is well worth a read.

Sample Web Server Applications Shipped with SIMPOL

The `\SIMPOL\Projects\ssp` directory contains a group of sample programs that demonstrate how to create and run applications from the web server. These include samples that are interlinked in some cases. It also includes both database and non-database related programs. The `sbiscalendar` project produces a simple calendar. The current day of the month is highlighted in a different color. There is also a useful tool for examining the environment variables that are available to the programming environment, called `sbisenvvars`. The following demonstrate basic database usage:

- `sbiscontact`
- `sbiscontactdisplay`
- `sbiscontactpost`
- `sbisreport`
- `sbisreportfast`

The starting point is the first item. To try it out at `http://www.simpol.com/cgi-bin/sbiscontact.smp`. These can all be set up to run on a local web server running on your development computer. The easiest approach is to install a local copy of the Apache web server. Then you will need to modify the local target in each project to match your system architecture. If the installer was able to find apache, it may have already modified the programs. The programs look for the `sbis.ini` file, so that may need to be modified, as will the `basehref` in the `header.htm` file. All of the items that are needed for the web server can be found in the `apache` inside of the `ssp` directory. It is a good idea to also run a local PPCS server to host the database. The samples ship with everything required in the `ppcsserver` directory in the project folder.

Finally there is a frame sample, which is admittedly someone outdated in terms of modern web design, but still may have some use. To run it, start the `sbisframesample.smp` in your browser using the correct URL or try it at `http://www.simpol.com/cgi-bin/sbisframesample.smp`.

The Web Server "Sandwich" Method

Many web server applications can be seen as the filling inside of a web page. The part of the web page that precedes the place in the content where the output of the program is placed is the top piece of bread, and the part that follows the content is the bottom piece of bread. In some cases it may be necessary to have 3 chunks of HTML that represent the page, the top part, from the initial HTML declaration down to the place in the header where the `basehref` is inserted, then the remainder of the header plus the start of the body, and finally the remainder of the body and the footer. In SIMPOL we use this approach quite heavily. A program can be made to look like a part of any target system by taking a typical page from that system, dividing it into the necessary chunks, and then using the `SBISInclude()` function to read those chunks of HTML from the storage media when the program is run on the web server. It is fast and efficient.

Web Server Application Summary

In this chapter we have discussed the development of web server applications using SIMPOL. We haven't gone into detail about how to actually program these, since the mechanics of programming a web server application are covered in two other places in the SIMPOL documentation, and a group of sample programs demonstrating various aspects of web server application programming are included with the sample projects. This is just the beginning of a journey. Web server programming is a large an growing topic, but by using this facility within SIMPOL you can feel confident that you can deploy web server applications that use databases stored in your system and that may even be used by desktop program in-house. We use this sort of approach ourselves, with a basic web system for use by our customers and an expanded desktop system written in SIMPOL for in-house use.

The tools are there, where you want to take it, is really up to you!

Chapter 10. Server Programs

About Server Programs

This chapter is meant to discuss the approach to building server programs. A server program may, or may not have a user-interface in the traditional sense. An example of a server program would be a program that accepts TCP/IP connections and then responded to requests by carrying out some process and then returning a result to the caller. A web server, a dedicated XML server that acts as a gateway to a database, a mail server, a payment gateway, a graphics server that converts images from one format to another, a file update system; these are all typical server programs.

Accessing Server Programs

There are a few different ways of accessing a server program. Most modern systems will use either TCP/IP, UDP/IP, a communications port (COM1, USB001, OLE2, DDE, etc.), the file drop method (monitor a directory and when a file appears in the directory read and process it), or they may use some other intermediary such as an ini file. Currently SIMPOL can be used to implement such systems using TCP/IP, file drop, or ini files (SIMPOL supports OLE2, but only as a client program).

Sample TCP/IP Server and Client Programs

Two sample programs are shipped with SIMPOL. They can be found in the `\SIMPOL\Projects\sockets` directory. One is called `clientsample` and the other is `serversample`. Together they implement a very rudimentary file transfer system that could fairly easily be built into a live update type of mechanism.

They each contain a `receivestring()` function that was taken directly from a commercial application built in-house. This particular function design requires that the other end respond with a carriage-return linefeed pair or a linefeed alone (should work fine on Windows, Linux, and OS-X). The function was designed so that it can also work together with directly typed input, such as from Telnet. The server simply sits there doing nothing until a connection is made. At that point a new thread is spawned and passed to the `dispatch()` function, along with the incoming socket object.

Within the `dispatch()` function a `HELO` is sent to identify the server as a response to the connection. A loop is entered to process the commands supported by the server. If nothing arrives within a specific amount of time, then the loop will time out and the connection will be closed. The client receives the `HELO` and then requests the time by sending the `TIME` command. After receiving the time from the server, it sends the `GET` command and the server responds by opening and then transmitting its own program file, prefaced with a minimal header to tell the client the amount of data being sent. The client receives the data using the `receiveblob()` function and stores it. It then sends the `QUIT` command to allow the server to exit the loop immediately rather than waiting for a time out.

Server Programs Conclusion

Although the sample program is quite basic, it is actually very powerful. With minor modifications it could easily send whichever file was requested. It could modify the header to send the file name, size, date and time of last modification, and even optionally send the data compressed and/or encrypted, using existing library code supplied with SIMPOL. With minor modifications it could also provide a query mechanism where the client sends a `LIST` command, for example, and it would then examine a designated directory and return the file name and last modified date and time for each file to the client. The client could then compare these with its own copy of each and request any file that was newer than the copy it already holds (plus any missing ones). Using the ability of the UTOSdirectoryentry object to set the date and time, the client could make sure that the local copy always has the same date and time as the server one. This sort of application could also be implemented as a web server program.

Chapter 11. Converting Legacy Superbase

Where to Begin?

Explaining how to convert from a product that is as multi-faceted as the legacy Superbase product is not an easy task. The number of different ways that people have used the product means that any detailed set of instructions will be certain to fail the needs of a large percentage of those interested. Instead, this chapter will discuss some guidelines and techniques for conversion.

It is probably easiest to start with what things *can* be converted fairly easily. That list is as follows:

- Database files (assuming they are not encrypted)
- Display forms (currently `DisplayTextBox` objects are not supported – rotated editable text boxes)
- Print forms (these are the same forms in legacy Superbase, but are primarily meant to be printed and are handled separately in Superbase NG)
- Dialog definitions (the `DialogFrame` object is not currently supported)
- Menu programs (as saved from the Superbase Menu Editor)
- Graphic Reports (some hand-tweaking may be required in the resulting XML)

What is noticeably lacking from all of the above is any mention of program code. Legacy Superbase supports three distinct styles of BASIC programming:

- Early QuickBasic with only global variables, and **GOTO**, **GOSUB**, and **RETURN**, with both line numbers and symbolic label names.
- Procedural BASIC with **SUB main()** local and global variables, user-defined functions, and an event handling mechanism for creating event-driven programs.
- Object BASIC with supplied objects for the GUI components, like forms, and form controls, dialogs and dialog controls.

That list mirrors a clear progression in the development of programming languages over the course of time. The problem is, legacy Superbase allowed the use of these different styles of programming concurrently. That isn't so bad for the final two, since the object BASIC is layered over the top of the procedural BASIC anyway. The problem is the original BASIC, and the excessive use of the **GOTO** and global variables.

There is nothing inherently *wrong* with the **GOTO** command (although some might argue very strongly about that), if it is used carefully (and sparingly) in the hands of a skilled programmer, but unfortunately it changes the direction of program execution permanently, and often cannot easily be followed by someone (or a program) reading the source code. Often the original author of the code will no longer understand how it works within even a few months of having written it.

Needless to say, if the original author no longer understands how their program works, the likelihood of any program understanding it, even one that is designed to convert source code, is very low. Having said that, it is not as bad as it sounds. Please read on.

How Superbase NG Differs

When SIMPOL was designed, one of the strongest factors in the design of the language was to make the code easy to learn, easy to use, and easy to maintain later. An unfortunately common scenario that has played out far too often in many places using tools like legacy Superbase, is where an island solution built by an inspired layman programmer achieves a degree of success. Then as its star rises, it requires additional professional assistance to make it to the next level of usability. At that point, the professionals investigate the software and discover that it is written in a way that is non-standard, complicated to understand, and possibly built using a tool that they personally have no experience using. At which point they decide to discard the original and start over again. The problem is the original solution probably took one dedicated person 1-3 years of work to build using a very powerful tool. The new version usually is estimated to require 3-5 people several years to produce, and would cost a fortune to achieve it. At which point the whole project might be scrapped as too expensive.

To prevent the solutions that were built by non-expert programmers from being discarded as unmaintainable or unsupportable once they reached this level, every effort was made to avoid this result. All the factors that were bars to entry for quickly learning the language were discarded. As much as possible, redundancy was removed from the design. The keyword set was reduced to the bare minimum and everything was turned into a type or a function. There is also no way to jump out of any block statement, so it is always clear how the code flows, and there are no global variables.

Not having global variables is one place where SIMPOL strongly differs from many languages. The choice to not allow them went back to the problems that are commonly associated with them:

- Random unexplained changes in one module as a result of calling code in some other module
- Assignment to apparent local variables changing the value of global variables
- The constant search for a new and valid name for a global variable
- The inability to distinguish in the code between a local variable and a global one

So what was the gain for SIMPOL by not having them, and how does it cope with certain situations that appear to *require* them? By not having global variables, all variable changes are specific to the function in which they are created. If a value is need in a function from outside the function, it *must* be passed into the function as a parameter. It is always clear where the values are coming from.

But what about event handlers? How do we get the data we need into an event handler if we don't call it directly? That is handled by every event having an additional property called reference. This property is declared to be of `type (*)`, which is a special placeholder that allows a variable to hold a reference to any data type. This is the mechanism used to pass quasi-global data to an event handling function. With that, the loop is closed and there is no other need for global variables.

Every SIMPOL program starts in the function `main()` and ends when that function is exited (unless the program is multi-threaded and one or more threads are still executing at that time).

That all sounds like loads of work, if there is a lot of legacy Superbase code to change. The reality is a little different. As it turns out, much of what people code is about working around how their environment works. Legacy Superbase is no different. The easy route is to move the data and the forms, migrate the menus, and then see what works and what is missing. Then add the code as event handlers for the various event types.

One important difference to note is that Superbase database files allow the definition of calculations, constants, and validations as part of the field definition. The initial Superbase NG database engine is a pure storage engine, and it does not cater for these field-level operations. This may seem to be a significant drawback, but in fact, most of the more advanced legacy Superbase developers had stopped using these in their projects quite some time ago, since in any complex project these sorts of things could get in the way and cause as much trouble as they provided help.

To resolve this in SIMPOL, it is necessary to migrate those settings into a function or set of functions. In an earlier chapter, Chapter 6, *GUI-Style Database Programs*, a special function was built to create the serial number when a new record is created in a table on a form. A similar function would be needed for each time a record is created in any table that needs constants to be generated at that time. A similar function would be needed for calculations, which could be called every time a record is saved (it could also be called during the onlostfocus event of certain controls).

So What's the *Good* News?

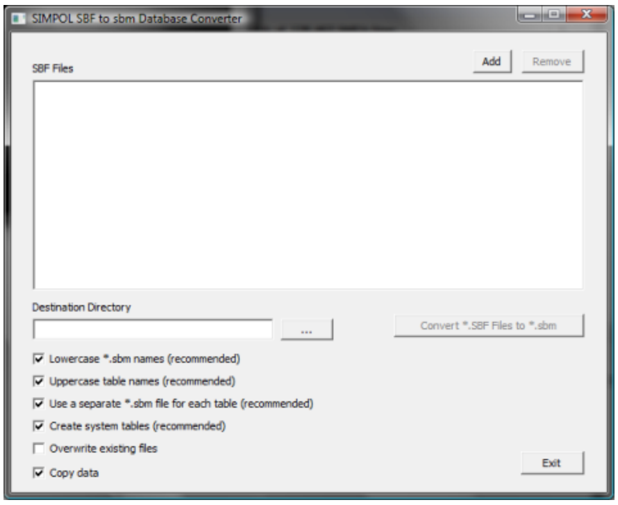
If the legacy Superbase content is primarily data and forms, with a few reports, the conversion should be pretty quick and painless. If there is a large amount of code, then the process can use the *phased migration* approach.

What *phased migration* means, is that there is a methodology where the data can be converted to use the new Superbase NG database format, the legacy Superbase application can be converted to use the PPCS method for accessing the data, and then over time, modules of the Superbase program can be converted into Superbase NG and called from the legacy Superbase program. Depending on the design of the legacy Superbase application, some items might be ready to convert sooner than others. Also, since both Superbase and Superbase NG can access data via the PPCS protocol, web server applications written in SIMPOL can be used to provide browser-based access to aspects of the converted Superbase data in real time. This capability to keep the original application in legacy Superbase and to gradually migrate it over the course of time is not available with other tools. It has the advantage that the existing software can be maintained and updated (wherever possible building new modules only in SIMPOL) and gradually more and more of it will actually be in SIMPOL. The key to this is that both can share the same database concurrently. Changing an existing legacy Superbase program to use PPCS instead of the normal method of accessing data rarely takes more than a single day, no matter how complicated the program is. PPCS was designed to be an easy move for legacy Superbase programmers. It *does* require the user to be on a fairly recent version of the legacy Superbase product, though. No less than version 3.6i, preferably as of build 496. Many of the supplied conversion tools need to run on the Superbase 2001 version or later.

Converting Superbase Databases to Superbase NG

There is a very useful tool supplied in the `Utilities` directory called `sbf2sbm.smp`, which converts legacy Superbase database files into Superbase NG's `*.sbm` format. This reads the data file directly, so it does not require any extra action to make it available, with one exception. It cannot read encrypted Superbase database files. In that case the file needs to be converted to a non-encrypted database file. Just as a note, currently there is no encrypted file format for Superbase NG database files. At the same time, since multi-user access is only via PPCS, the location of the physical data does not need to be accessible to every user as is the case with the older Superbase LAN and Distributed LAN networking.

Running the SBF2sbm tool presents a dialog window like the one below:


SIMPOL SBF to sbm Database Converter

SBF Files

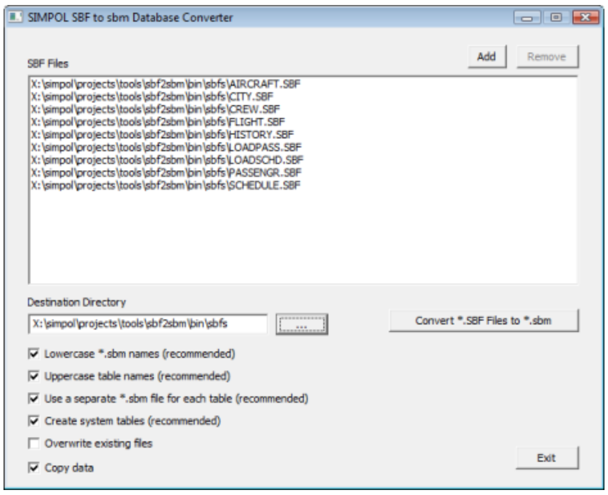
Destination Directory

☑ Lowercase *.sbm names (recommended)
☑ Uppercase table names (recommended)
☑ Use a separate *.sbm file for each table (recommended)
☑ Create system tables (recommended)
☐ Overwrite existing files
☑ Copy data

Image of the SBF2sbm dialog.

As an experiment, we are going to import the database tables from the Superbase Air example that ships with all versions of Superbase 3.x (Superbase 95, Superbase 3.01, Superbase 3.02, Superbase 3.2, Superbase 3.5, Superbase 3.6i, Superbase 2001, and Superbase Classic).

To start with, we click on the Add button, which let's us select the `*.sbm` files and add them to the list for conversion. The dialog supports multiple selection, so we can select all the files at once. Then click on the … button to select the target directory (by default it will be set to the same as the most recent source directory).


SIMPOL SBF to sbm Database Converter

SBF Files

```
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\AIRCRAFT.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\CITY.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\CREW.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\FLIGHT.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\HISTORY.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\LOADPASS.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\LOADSCHD.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\PASSENGR.SBF
X:\simpol\projects\tools\sbf2sbm\bin\sbfs\SCHEDULE.SBF
```

Destination Directory: X:\simpol\projects\tools\sbf2sbm\bin\sbfs

- ☑ Lowercase *.sbm names (recommended)
- ☑ Uppercase table names (recommended)
- ☑ Use a separate *.sbm file for each table (recommended)
- ☑ Create system tables (recommended)
- ☐ Overwrite existing files
- ☑ Copy data

Image of the SBF2sbm dialog ready to convert.

Leave the settings at their default values for the most successful conversion. Below each of the settings is explained.

- Lowercase *.sbm names (recommended) — This makes sure that if the tables are being converted into separate container files, one per Superbase file, that the file names will be forced to lowercase, otherwise they will be in uppercase (like the original files from Superbase).
- Uppercase table names (recommended) — This ensures that the tables are created with uppercase names. This is important if working on a hybrid solution, since SIMPOL is case-sensitive when opening the tables.
- Use a separate *.sbm file for each table (recommended) — Although SIMPOL database containers can support multiple tables, there are good, performance related reasons for keeping each table in a separate container. It also makes it easier when doing updates of specific tables, or if reorganizing only one table.
- Create system tables (recommended) — Unlike Superbase, SIMPOL database fields only have a data type, and whether they are indexed or not (plus if they are, an index algorithm and precision). Things like the display format are not part of the core field definition, but the system tables store additional information such as the display format, help string, share name (which can be different to the field name) and other useful bits. Using the system tables means that the standard PPCS server program can automatically share the table and have it look just like the original from Superbase (minus calculations, etc.).
- Overwrite existing files — If selected, it overwrites an existing file without asking. If it is not selected, it will not ask, and will not overwrite.
- Copy data — This determines if the table is created empty, or if all the data is also transferred.

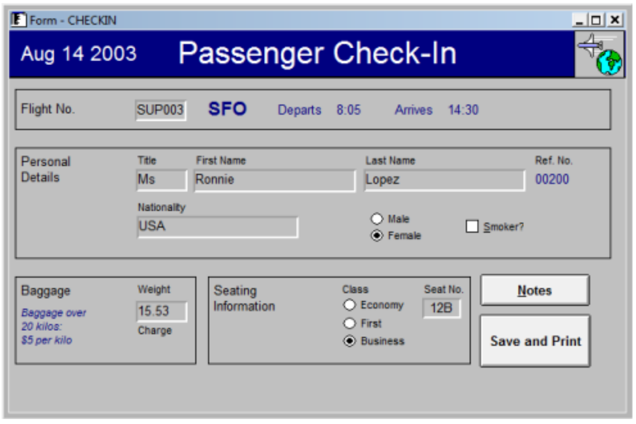
Note

One thing that it is important to understand, is that this tool cannot resolve the calcula-tions for a virtual field. If the table definition has virtual calculated fields, and if those fields are unique indexes, then the import of that table will fail. This was the case with the `SCHEDULE.SBF`, since it turned out to have multiple virtual calculated fields, one of which had a unique index on it. In order to successfully import the table, the fields need to be changed to normal fields (not virtual), and the content needs to be updated by doing a Superbase **UPDATE** that specifically sets each of these fields to be equal to itself.

Converting the Forms

Now that the data is in Superbase NG format, we can convert the forms. The form conversion tool is a SIMPOL program. This program reads the `*.sbv` files directly. It is also the only way to recover embedded images that are in the form itself. To run this tool, from the Start menu, select Superbase NG → Superbase Classic Conversion Tools. This will launch the tool that hosts the various Superbase conversion tools. Select the one for converting the forms. It will look very similar to the one we used for converting databases. Select the form(s) you wish to convert and provide a target directory. Click on the Convert Forms button and the forms will be converted into the target directory.

In this example, we will be converting the `CHECKIN.SBV`. The original is shown below.



Form - CHECKIN

Aug 14 2003 Passenger Check-In

Flight No. SUP003 SFO Departs 8:05 Arrives 14:30

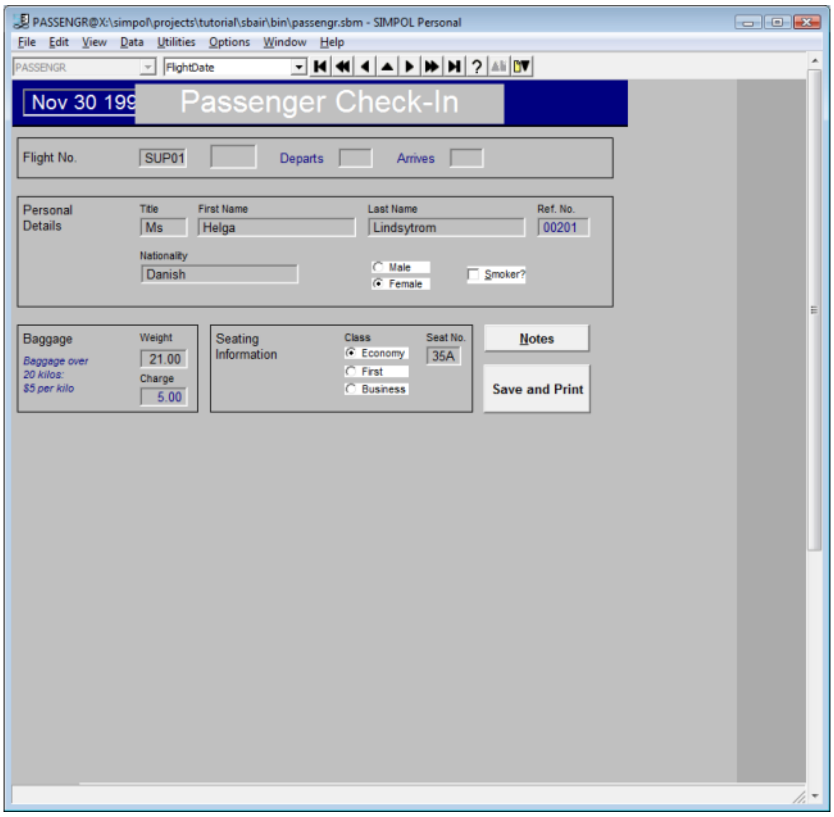
Personal Details
Title: Ms
First Name: Ronnie
Last Name: Lopez
Ref. No.: 00200
Nationality: USA
Male [● Female]
☐ Smoker?

Baggage
Weight: 15.53
Charge
Baggage over 20 kilos: $5 per kilo

Seating Information
Class: Economy / First / [● Business]
Seat No.: 12B

Image of the CHECKIN form prior to conversion.

The following image is of the converted form, without any modifications done to it, merely opened as is in Superbase NG Personal.


PASSENGR@X:\simpol\projects\tutorial\sbair\bin\passengr.sbm - SIMPOL Personal

Nov 30 199
Passenger Check-In

Flight No. SUP01 Departs Arrives

Personal Details
Title: Ms
First Name: Helga
Last Name: Lindsytrom
Ref. No.: 00201
Nationality: Danish
Male [✓ Female]
Smoker?

Baggage
Baggage over 20 kilos: $5 per kilo
Weight: 21.00
Charge: 5.00

Seating Information
Class: [✓ Economy] First Business
Seat No.: 35A

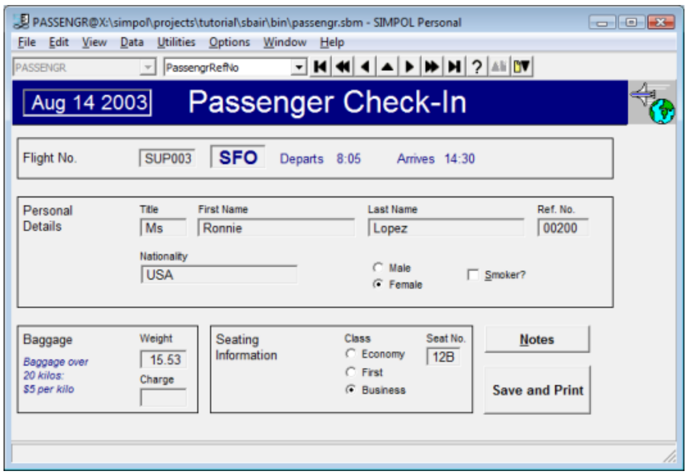
Image of the CHECKIN form after conversion.

As we can see from the converted form, it looks different in a number of ways. That is in part because the controls on a Superbase form are not native controls. Instead they are all drawn by Superbase. Also, the original form was not sized correctly, it is actually much longer than it needs to be. Also, in Superbase the background of a label could be optionally turned off. This is not an option with real Windows controls. The rest are relatively minor adjustments. One thing that the conversion program does not yet do, is transfer the tab order. That is because tab order works quite differently between the two. Eventually this may also be added to the converter. That was hand adjusted as part of the work in the Form Designer.

Another thing that is different is the Superbase text boxes in the original form that have no border, and are not recessed. This option is not available in standard Windows controls. SIMPOL has its own trick here though: dataform1text controls are data-aware (can be bound to a field), labels in Superbase are not. Also note that the buttons are using a different color scheme to the rest. That is because they were defined as using the standard color scheme. In Superbase the colors are fixed. In SIMPOL, they can be tied to the current theme.

Another problem is the missing image. The reason it is missing is that the image was pasted into the form from the clipboard, and there is no way to extract the image from the form programmatically. In fact, the only way to do it is to capture it off the form.

By opening the form in Superbase NG Personal and modifying it in the SIMPOL Form Designer, the resulting form looks like this. This is being shown in Superbase NG Personal.


PASSENGR@X:\simpol\projects\tutorial\sbair\bin\passengr.sbm - SIMPOL Personal
File Edit View Data Utilities Options Window Help
Notes
Save and Print

PASSENGR | PassengrRefNo

Aug 14 2003 Passenger Check-In

Flight No. SUP003 SFO Departs 8:05 Arrives 14:30

Personal Details: Title Ms; First Name Ronnie; Last Name Lopez; Ref. No. 00200; Nationality USA; ○ Male [✓ Female]; ☐ Smoker?

Baggage: *Baggage over 20 kilos: $5 per kilo*; Weight 15.53; Charge

Seating Information: Class ○ Economy ○ First [✓ Business]; Seat No. 12B

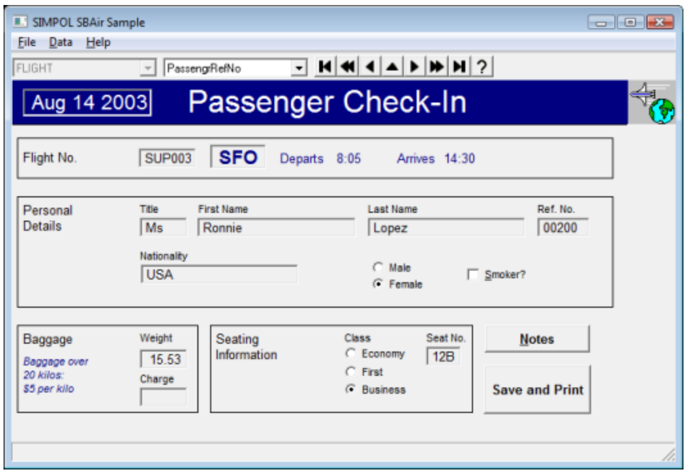
Image of the CHECKIN form after repair in the Form Designer.

As we can see, it now looks very much like the original. In some ways, it looks better. It now uses the system theme for the vast majority of colors (not all, however, since some text was blue, and it is not supported to have the text color using system colors and the background color using fixed colors). The information in the Flight No. section of the form that is in blue is actually looked up from the `FLIGHT` table using the `FlightNo` as the link.

Creating the Application

The steps to turn this into an application of its own are quite simple really. As described in the section called “Summary”, all it really takes is to create a project, which I called `sbair`. I then copied all the program files from the `addressbook` program into the new project directory, renaming the one called `addressbook.sma` to `sbair.sma`. I also copied all the toolbar images from the `addressbook\bin` directory into the `sbair\bin` directory. Into the same directory I copied the converted database files (at this point we only need `passengr.sbm` and `flight.sbm`), and the converted and reworked form: `checkin.sxf`.

In terms of changes to the program code, all the suggestions made in the section called “Summary” were applied. The resulting program came up immediately and can be seen below. The copying and code changes took less than ten minutes, including compiling and fixing things that were forgotten.



FLIGHT | PassengrRefNo

Aug 14 2003 Passenger Check-In

Flight No. SUP003 SFO Departs 8:05 Arrives 14:30

Personal Details: Title Ms; First Name Ronnie; Last Name Lopez; Ref. No. 00200; Nationality USA; ○ Male [✓ Female]; ☐ Smoker?

Baggage: *Baggage over 20 kilos: $5 per kilo*; Weight 15.53; Charge

Seating Information: Class ○ Economy ○ First [✓ Business]; Seat No. 12B

Image of the CHECKIN form in the new SB Air application.

This is, of course, just the beginning. A full conversion would convert each of the forms, add navigation to the menu, add functions to support the buttons on the forms, add functions to handle constants and calculations for the tables, etc. The goal here was only to demonstrate the approach, not do a full conversion.

Summary

In this chapter we discussed the issues facing a conversion from Superbase to SIMPOL. We also looked at various scenarios and discussed why SIMPOL offers the easiest conversion solution for existing Superbase projects, especially the advantage of doing a *phased conversion* where over the conversion period there is a hybrid Superbase/SIMPOL application that starts out wholey in Superbase and eventually bit by bit becomes completely SIMPOL.

Then using the tools, we converted a portion of a standard sample shipped as part of the Superbase 3.x series, including converting the database tables and one form. That form we then cleaned up in the SIMPOL Form Designer and finally we built a standalone application for it in just a few minutes by stealing most of the code from a standard sample program.