Safire

Safire Language Reference

End User and Software Developer Guide

Source-first language guide

Applications, projects, data declarations, procedures, classes, database dictionaries, queries, windows, controls, events, reports, services, packaging, diagnostics, and runtime objects.

Designed for end users, application owners, and software developers building long-lived business systems.

Safire Documentation
Version date: 2026-07-01
Readable source. Visual productivity. Deployable applications.

Contents

1 Safire Language Reference

Document status: Current Safire language reference guide for end users, application designers, software developers, support teams, and technical decision makers.

Audience:

- End users and support staff who need to understand how Safire applications are organized, configured, validated, packaged, and supported.
- Software developers who write Safire source, define data dictionaries, create windows, bind controls, handle events, build reports, call runtime services, and package applications.
- Application designers who work visually while preserving readable source.
- Technical owners who need a practical reference for long-lived business systems.

Scope: This guide describes Safire source files, project structure, declarations, data types, expressions, procedures, classes, dictionary and database declarations, queries, windows, controls, events, data binding, reports, runtime objects, resources, configuration, packaging, diagnostics, and language reference entries.

Safire source rule: Safire source files are the durable truth of the application. Visual tools, source editors, report designers, data designers, application tools, validation, diagnostics, packaging, runtime services, and approved automation must read from and write back to the same source-owned project model.

2 How to read this guide

This guide is written as a practical reference rather than as a tutorial only. Each language item is presented in a repeatable format.

Section item	Meaning
Syntax	The source shape used by the declaration or statement.
Purpose	What the item does in a Safire application.
Parameters or properties	Values supplied by the developer.
End user effect	How the item affects the running application.
Developer usage	How to declare, organize, and maintain the item.
Comments	Practical design guidance.
Example	A short source example.

The examples use a clear lowercase style for keywords. Project tools may preserve user formatting, but the examples in this guide use one consistent style so that source remains easy to scan.

A source example is not a complete application unless the section says it is complete. Real projects normally split source into application, windows, reports, dictionaries, queries, classes, procedures, resources, and project manifests.

3 Core product ideas

3.1 Source-first application truth

A Safire application is defined by source files and project manifests. Generated output, previews, logs, caches, indexes, temporary files, and packaged output are reproducible artifacts. They help the developer, but they are not the primary application truth.

```
application CustomerManager
  title "Customer Manager"
  version "1.0.0"
  mainWindow CustomerListWindow
end
```

3.2 Readable business source

Safire source is intended to read like a business application model. It names applications, tables, fields, windows, controls, events, reports, procedures, classes, services, resources, and packaging information directly.

3.3 Visual and textual editing

A Safire project can be edited visually and textually. The visual designer must not be the only place where meaning exists. When a window, report, dictionary, or query is edited visually, the change must be represented in source-owned project data.

3.4 Runtime objects

Safire runtime services are exposed as named language objects and procedures. A developer should be able to see and use the same object names in source, diagnostics, documentation, and application tools.

3.5 Deployment without design tools

A deployed application contains the application and the runtime components needed to execute it. End users should not need the design environment to run the finished application.

4 Current capability baseline

This guide reflects the current documented Safire product surface. Some features are source-level declarations, some are runtime object contracts, and some are complete application patterns. The important user-facing rule is that the public names, source shapes, and application concepts remain stable and understandable.

Area	Safire capability
Project model	Solution and project manifests, source folders, resources, build/run/package metadata, and support diagnostics.
Language core	Modules, declarations, variables, expressions, control flow, procedures, functions, classes, structured types, conversions, nullable values, arrays, lists, and maps.
Standard runtime objects	Files, paths, strings, dates, structured data, diagnostics, result status, runtime errors, documents, reports, communication objects, and service objects.
Database system	Database, table, field, key, index, relationship, constraint, transaction, query, migration, local store, server store boundary, and database diagnostics.
Desktop UI	Applications, windows, panels, labels, entry fields, buttons, browse lists, grids, menus, toolbars, status bars, events, validation, bindings, and layout overlays.
Reports	Report definitions, page setup, bands, groups, totals, preview, print, export, and data-bound report elements.
Web and service apps	Routes, request and response objects, secure service configuration, static assets, application hosting, sessions, authentication, logging, health checks, and support bundles.
Security objects	Certificates, keys, secure contexts, secure sockets, secure client/server options, certificate store, renewal, mutual authentication policy, and secure diagnostics.
Packaging	Windows application artifacts, runtime libraries, configuration, resources, database files, report assets, diagnostics, and support material.

5 Project and source organization

5.1 Typical project tree

```
CustomerManager/
  solution.sfsol
  project.sfproj
  app/
    Application.sf
  windows/
    CustomerList.sfwin
    CustomerEdit.sfwin
  reports/
    CustomerList.sfrpt
    InvoiceList.sfrpt
  dictionary/
    Customer.sfdict
    Invoice.sfdict
  queries/
    CustomerSearch.sfquery
    InvoiceList.sfquery
  classes/
    CustomerService.sf
    InvoiceService.sf
  procedures/
    CommonProcedures.sf
  resources/
    icons/
    images/
    strings/
  config/
    appsettings.json
  build/
  output/
```

5.2 Project manifest

The project manifest names the project, project type, main source file, source folders, resource folders, output settings, and application artifact type. It is meant to be readable and reviewable.

```
{
 "schemaVersion": 2,
 "projectName": "CustomerManager",
 "projectType": "desktop-app",
 "mainSource": "app/Application.sf",
 "sources": [
  "app/Application.sf",
  "windows/CustomerList.sfwin",
  "windows/CustomerEdit.sfwin",
  "dictionary/Customer.sfdict",
  "queries/CustomerSearch.sfquery"
 ],
 "resources": ["resources"],
 "artifactType": "windows-exe",
 "subsystem": "windows-gui"
}
```

5.3 Solution manifest

The solution manifest groups related projects. A business system may contain a desktop application, service application, report package, shared library, import tool, and test or diagnostic project.

```
{
  "schemaVersion": 1,
  "solutionName": "CustomerSuite",
  "projects": [
    "CustomerManager/project.sfproj",
    "CustomerService/project.sfproj",
    "SharedBusiness/project.sfproj"
  ]
}
```

5.4 Source module types

Module type	Usual file extension	Purpose
Application	.sf	Application title, startup, main window, resources, default data, application settings.
Window	.sfwin	Source-backed form, screen, dialog, browse window, or utility window.
Report	.sfrpt	Printable or exportable report definition.
Dictionary	.sfdict	Database, table, field, key, relationship, and validation metadata.
Query	.sfquery	Reusable data selection and sorting definition.
Class	.sf	Business object, service, rule set, helper, runtime wrapper, or reusable component.
Procedure	.sf	Standalone executable logic.
Resource	.sfres	Named resources such as icons, images, strings, help, and document assets.
Config	.json	Configuration defaults and deployment settings.

5.5 Generated output rule

Generated files may exist to speed development, support previews, or package runtime output. A generated file is not application truth unless it is deliberately promoted into source and maintained as part of the project.

6 Application declaration and startup

The application declaration describes the top-level program. It normally includes title, version, publisher, main window, default database, resources, culture, and startup behavior.

6.1 Syntax

```
application Name
   property value
   property value
end
```

6.2 Example

```
application SalesManager
   title "Sales Manager"
   version "1.0.0"
   company "Example Trading"
   mainWindow SalesMainWindow
   defaultDatabase MainData
   icon Resource("icons/app.ico")
   culture "en-ZA"
end

procedure Main()
   LoadConfiguration()
   OpenWindow(SalesMainWindow)
end
```

6.3 Common application properties

Property	Type	Purpose
title	String	Display name of the application.
version	String	Application version shown in about boxes, support bundles, and package metadata.
company	String	Publisher, organization, or product owner.
mainWindow	Window reference	First user-facing window opened by the application.
defaultDatabase	Database reference	Default database or dictionary root.
icon	Resource reference	Application icon.
culture	String	Default language, region, and formatting culture.
licenseMode	String or enum	Optional application licensing mode.
startupMode	Enum	Window, service, console, import, maintenance, or test startup behavior.

6.4 End user effect

The application declaration controls what the user sees first: application name, icon, first window, default language, startup behavior, and runtime package identity.

6.5 Developer usage

Keep application declarations short. Put business rules in procedures or classes. Put user interface structure in windows. Put data shape in dictionaries. Put report output in report files. Put deployable assets in resource declarations.

7 Source code format

7.1 Statements

A statement normally occupies one line. Prefer one logical action per line.

```
CustomerName = FirstName + " " + Surname
Balance += InvoiceTotal
Refresh(CustomerBrowse)
```

7.2 Blocks

Compound declarations and execution structures are terminated with `end`.

```
if Balance > CreditLimit then
   ShowWarning("Customer is over the credit limit.")
end
```

7.3 Comments

```
// Import orders captured while the branch was offline.
ImportOfflineOrders()

/*
Validation is separated from posting so that all user-facing
messages can be shown before any records are committed.
*/
ValidateBatch()
```

7.4 Documentation comments

Documentation comments describe public procedures, functions, classes, tables, reports, and windows. Tools may use them for reference documentation, help panels, and assistant context.

```
/// Posts an invoice after validation and ledger preparation.
/// Returns a result status with user-facing error text when posting fails.
function PostInvoice(InvoiceId is Integer) returns ResultStatus
   return InvoiceService.Post(InvoiceId)
end
```

7.5 Line continuation

Long calls should be split across lines at natural boundaries. Prefer clear indentation over very long statements.

```
OpenWindow(CustomerEditWindow,
   mode = Change,
   customerId = CustomerBrowse.CurrentRow.CustomerId)
```

8 Identifiers and naming

Identifiers name source objects. Use stable business names. Avoid names that describe temporary implementation details.

Object	Recommended pattern	Example
Application	Business name	CustomerManager
Window	Purpose + Window	CustomerListWindow
Report	Purpose + Report	InvoiceListReport
Query	Purpose + Query	CustomerSearchQuery
Button	Action + Button	SaveButton
Label	Field + Label	CustomerNameLabel
Entry field	Field + Entry	CustomerNameEntry
Browse	Entity + Browse	CustomerBrowse
Class	Business service or object	CustomerService
Procedure	Action verb	PostInvoice
Field	Business term	CustomerName

8.1 Identifier rules

A Safire identifier should start with a letter and should contain letters, digits, and underscore characters. Keep names clear in source and in the designer. Avoid reserved words.

8.2 Qualified names

```
Customer.Name = "Acme Trading"
Invoice.CustomerId = Customer.CustomerId
CustomerService.Save(Customer)
CustomerBrowse.CurrentRow.CustomerId
```

9 Data declarations and types

9.1 Variables

```
var CustomerName is String(100)
var Quantity is Integer = 1
var UnitPrice is Money = 0.00
var IsActive is Boolean = True
var PostedAt is DateTime = Now()
```

9.2 Constants

```
const DefaultVatRate is Decimal = 15.00
const CompanyName is String = "Example Trading"
```

9.3 Common scalar types

Type	Purpose	Example
Boolean	True or False values.	True
Integer	Standard whole number.	12345
Long	Large whole number.	1234567890
Decimal	Fixed precision decimal value.	123.45
Money	Currency and business money amounts.	99.95
Real	Floating point measurement value.	3.14159
String(n)	Text with maximum length.	String(100)
Text	Long text or memo content.	Notes
Date	Calendar date.	Today()
Time	Time of day.	14:30:00
DateTime	Date and time.	Now()
Guid	Globally unique identifier.	NewGuid()
Blob	Binary content such as file data.	DocumentBytes
Any	Controlled dynamic value for generic routines.	Value

9.4 Strings

```
CustomerName = "Acme Trading"
Message = "Invoice posted successfully."
PathText = "data\customers.dat"
```

Escape	Meaning
\\n	New line.
\\t	Tab.
\\"	Double quote.
\\\\	Backslash.

9.5 Dates, times, decimals, and money

Date, time, decimal, and money values should remain typed values. Do not store financial values as ordinary text when calculation, sorting, or validation is required.

```
var InvoiceDate is Date = Today()
var PostedAt is DateTime = Now()
var VatAmount is Money
VatAmount = Round(ExclusiveAmount * VatRate / 100, 2)
```

9.6 Nullable values

Nullable values allow a field or variable to contain no value. Use nullable values for optional business data, not as a replacement for validation.

```
var DeliveryDate is Date?

if DeliveryDate is null then
    ShowWarning("Delivery date has not been captured.")
end
```

9.7 Arrays, lists, and maps

```
var Scores is Array<Integer>(10)
var CustomerNames is List<String(100)>
var Settings is Map<String, String>

CustomerNames.Add("Acme Trading")
Settings["Theme"] = "Light"
```

9.8 Structured records

```
type CustomerSummary
    CustomerId is Integer
    Name is String(100)
    Balance is Money
end

var Summary is CustomerSummary
Summary.Name = "Acme Trading"
```

9.9 Enumerations

```
enum InvoiceStatus
    Draft
    Posted
    Cancelled
end

var Status is InvoiceStatus = InvoiceStatus.Draft
```

10 Expressions and operators

10.1 Assignment

```
CustomerName = "Acme Trading"
Quantity = Quantity + 1
Balance += InvoiceTotal
```

Operator	Meaning
=	Assign value.
+=	Add and assign.
-=	Subtract and assign.
*=	Multiply and assign.
/=	Divide and assign.

10.2 Arithmetic

```
LineTotal = Quantity * UnitPrice
InvoiceTotal = SubTotal + VatAmount
AverageDays = TotalDays / InvoiceCount
Remainder = ItemCount % PackSize
```

10.3 Comparison

```
if Balance > CreditLimit then
   ShowWarning("Credit limit exceeded.")
end

if CustomerCode = "CASH" then
   AllowCredit = False
end
```

Operator	Meaning
= or ==	Equal comparison in conditions.
<> or !=	Not equal.
<	Less than.
<=	Less than or equal.
>	Greater than.
>=	Greater than or equal.

10.4 Logical expressions

```
if IsActive and Balance > 0 then
   SendStatement(CustomerId)
end

if not CustomerExists(CustomerId) then
   ShowError("Customer not found.")
end

if PaymentMethod = "Cash" or PaymentMethod = "Card" then
   MarkAsPaid()
```

```
end
```

10.5 String expressions

```
FullName = FirstName + " " + Surname
DisplayText = CustomerCode + " - " + CustomerName
```

10.6 Function calls in expressions

```
VatAmount = CalculateVat(ExclusiveAmount, VatRate)
CustomerName = Trim(CustomerName)
ReportTitle = Upper(CompanyName) + " CUSTOMER LIST"
```

11 Execution control

11.1 If

```
if Balance > CreditLimit then
   StatusText = "Over limit"
elseif Balance > 0 then
   StatusText = "Open balance"
else
   StatusText = "Clear"
end
```

11.2 Case

```
case PaymentMethod
   when "Cash"
      PostCashPayment()
   when "Card"
      PostCardPayment()
   when "Transfer"
      PostBankPayment()
   else
      ShowError("Unknown payment method.")
end
```

11.3 For each loop

```
loop for each Line in InvoiceLines
   SubTotal += Line.LineTotal
end
```

11.4 Counter loop

```
loop i from 1 to 10
   Trace("Line " + ToString(i))
end
```

11.5 While loop

```
loop while Reader.HasMoreRows()
   ImportRow(Reader.CurrentRow)
   Reader.MoveNext()
end
```

11.6 Break, continue, and return

```
loop for each Line in InvoiceLines
   if Line.Cancelled then
      continue
   end
```

```
    if Line.ProductId = 0 then
        break
    end
end

function IsValidCustomer(CustomerId is Integer) returns Boolean
    if CustomerId <= 0 then
        return False
    end
    return CustomerExists(CustomerId)
end
```

12 Procedures and functions

12.1 Procedure

A procedure performs work and does not return a value.

```
procedure PostInvoice(InvoiceId is Integer)
   ValidateInvoice(InvoiceId)
   SaveLedgerEntries(InvoiceId)
   MarkInvoicePosted(InvoiceId)
end
```

12.2 Function

A function performs work and returns a value.

```
function CalculateVat(Amount is Money, Rate is Decimal) returns Money
   return Round(Amount * Rate / 100, 2)
end
```

12.3 Parameters

```
procedure SendStatement(CustomerId is Integer,
                        EmailAddress is String(255))
   // Send the customer statement.
end
```

12.4 Optional parameters

```
procedure ShowMessage(Message is String,
                      Title is String = "Safire")
   MessageBox(Title, Message)
end
```

12.5 Reference parameters

```
function TryGetCustomerName(CustomerId is Integer,
                            ref Name is String) returns Boolean
   if not CustomerExists(CustomerId) then
      return False
   end

   Name = CustomerNameFor(CustomerId)
   return True
end
```

12.6 Result objects

When a routine can fail for a business or runtime reason, prefer returning a result object instead of only returning True or False.

```
function PostInvoice(InvoiceId is Integer) returns ResultStatus
   if InvoiceId <= 0 then
      return ResultStatus.Failed("Invoice number is required.")
   end

   // Posting logic.
   return ResultStatus.Ok()
end
```

12.7 Developer comments

Keep event handlers short. Place business rules in procedures, functions, classes, and validation declarations. This makes windows easier to maintain and easier to test.

13 Classes and structured types

13.1 Class

```
class CustomerService
   private Db is DatabaseConnection

   public procedure Open()
      Db = OpenDatabase(MainData)
   end

   public function FindName(CustomerId is Integer) returns String
      return CustomerNameFor(CustomerId)
   end
end
```

13.2 Visibility

Keyword	Meaning
public	Available to other modules that can see the class.
private	Available only inside the class or module.
protected	Available inside the class and derived classes.
internal	Available inside the project or package.

13.3 Business service pattern

```
class InvoiceService
   public function Post(InvoiceId is Integer) returns ResultStatus
      ValidateBeforePost(InvoiceId)
      WriteLedgerEntries(InvoiceId)
      MarkInvoicePosted(InvoiceId)
      return ResultStatus.Ok()
   end

   private procedure ValidateBeforePost(InvoiceId is Integer)
      // Business validation belongs here, not in a button only.
   end
end
```

13.4 Construction and lifetime

```
var Service is InvoiceService
Service.Open()
Result = Service.Post(CurrentInvoiceId)
```

14 Error handling and validation

14.1 Try and catch

```
try
    PostInvoice(InvoiceId)
catch ex is Exception
    ShowError(ex.Message)
end
```

14.2 Throw

```
if InvoiceId <= 0 then
    throw New ValidationException("Invoice number is required.")
end
```

14.3 Validation declaration

```
validation CustomerRules for Customer
    rule NameRequired
        when IsBlank(Customer.Name)
        message "Customer name is required."
    end

    rule CreditLimitPositive
        when Customer.CreditLimit < 0
        message "Credit limit may not be negative."
    end
end
```

14.4 Using validation

```
if not Validate(Customer) then
    ShowValidationMessages()
    return
end

Save(Customer)
```

14.5 Validation result

```
Result = ValidateWithResult(Customer)
if Result.Failed then
    ShowError(Result.Error.Message)
    return
end
```

14.6 Comments

Validation is part of the source model. A form, service, report, import routine, and application tool should be able to understand the same rule names and user-facing messages.

15 Dictionary and database declarations

Safire treats database-backed business systems as a primary application type. Database structures should be visible in source rather than hidden inside unrelated routines.

15.1 Database

```
database MainData
    provider "LocalStore"
    databaseName "data/customer_manager.sdb"
end
```

Property	Purpose
provider	Database provider or store type.
databaseName	Database file, catalog, or logical name.
connectionName	Named connection from configuration.
userName	Optional user name from configuration.
password	Protected secret reference.
mode	ReadWrite, ReadOnly, Maintenance, or Import mode.

15.2 Table

```
table Customer
    field CustomerId is Integer primaryKey autoNumber
    field CustomerCode is String(20) required unique
    field Name is String(100) required
    field EmailAddress is String(255)
    field CreditLimit is Money default 0.00
    field Balance is Money default 0.00
    field IsActive is Boolean default True

    key CustomerCodeKey = CustomerCode unique
    key CustomerNameKey = Name
end
```

15.3 Field properties

Property	Meaning
required	Field must contain a value.
default	Default value for a new record.
primaryKey	Main record identity.
autoNumber	Value generated by the store or runtime.
unique	Duplicate values are not allowed.
displayName	User-facing field label.
picture	Display or input formatting rule.
lookup	Links to a lookup table or list.
validate	Attaches a validation rule.
readOnly	Value can be displayed but not edited in ordinary entry forms.

15.4 Keys and relationships

```
key CustomerCodeKey = CustomerCode unique
```

```
key ActiveNameKey = IsActive, Name

table Invoice
  field InvoiceId is Integer primaryKey autoNumber
  field CustomerId is Integer required
  field InvoiceDate is Date required
  field TotalAmount is Money default 0.00

  relation CustomerLink
    from CustomerId
    to Customer.CustomerId
    onDelete Restrict
  end
end
```

15.5 Constraints

```
constraint CreditLimitRange for Customer
  when Customer.CreditLimit < 0 or Customer.CreditLimit > 1000000
  message "Credit limit is outside the allowed range."
end
```

15.6 Transactions

```
transaction PostInvoiceTransaction
  begin
    Save(Invoice)
    Save(LedgerEntries)
    MarkInvoicePosted(Invoice.InvoiceId)
  commit
  onError
    rollback
    ShowError("Invoice could not be posted.")
  end
end
```

15.7 Database runtime objects

Object	Purpose
DatabaseConnection	Open, close, and manage a named database connection.
TableObject	Represents a logical table and row access.
FieldObject	Represents field name, type, required settings, defaults, and display metadata.
KeyObject	Represents primary and candidate keys.
IndexObject	Represents lookup and sorting indexes.
RelationshipObject	Represents parent-child table relationships.
ConstraintObject	Represents required, unique, range, relationship, and custom data rules.
TransactionObject	Represents begin, commit, rollback, and transaction diagnostics.
QueryObject	Represents filter, sort, projection, and join execution contract.
MigrationObject	Represents schema version movement and diagnostics.
LocalStore	Represents local durable table storage.
ServerStore	Represents server-backed store and synchronization

	boundary.
DbDiagnostic	Represents schema, data, query, and transaction error information.

16 Queries, views, and data selection

16.1 Query declaration

```
query CustomerSearchQuery
   parameters
      SearchText is String(100) = ""
      ActiveOnly is Boolean = True
   end

   from Customer
   where Customer.Name contains SearchText
   order by Customer.Name
end
```

16.2 Using a query

```
Rows = ExecuteQuery(CustomerSearchQuery,
   SearchText = SearchText,
   ActiveOnly = True)
```

16.3 Query columns

```
query CustomerBalanceQuery
   parameters
      MinimumBalance is Money
   end

   from Customer
   select CustomerId, CustomerCode, Name, Balance
   where Balance >= MinimumBalance
   order by Name
end
```

16.4 Query comments

Use query declarations when the same data selection is needed by a browse, report, export, background job, or business service. This avoids scattering repeated data selection logic through windows and procedures.

17 Windows, forms, and controls

Windows are source-backed Safire application artifacts. A visual designer can edit them, but the source-owned window definition remains authoritative.

17.1 Window declaration

```
window CustomerListWindow
  title "Customers"
  bounds 40, 40, 900, 600
  startupPosition CenterScreen
  dataSource Customer

  browse CustomerBrowse
    bounds 10, 45, 860, 460
    rows CustomerSearchQuery(SearchText = SearchText)
    column CustomerCode title "Code" width 90
    column Name title "Customer" width 260
    column Balance title "Balance" width 100 align Right format "money"
  end

  button NewButton
    text "New"
    bounds 10, 520, 80, 28
    on Click call NewCustomer
  end

  button CloseButton
    text "Close"
    bounds 790, 520, 80, 28
    on Click call CloseWindow
  end
end
```

17.2 Common window properties

Property	Type	Purpose
title	String	Window title shown to the user.
bounds	Rectangle	X, Y, width, and height.
startupPosition	Enum	Default, center screen, center parent, saved position.
visible	Boolean	Whether the window is initially visible.
enabled	Boolean	Whether the window accepts input.
resizable	Boolean	Whether the user can resize the window.
minSize	Size	Minimum width and height.
icon	Resource	Window icon.
dataSource	Data reference	Default data source for binding.
layout	Layout reference	Sizer, anchor, dock, grid, or absolute layout information.

17.3 Control catalog summary

Control	Typical use

window	Top-level screen, dialog, form, browse, or report shell.
panel	Container for related child controls.
label	Static text, field caption, or instruction.
entry	Text, numeric, date, or other single-value input.
button	Command or action.
checkbox	Boolean choice.
radioButton	Exclusive choice within a group.
comboBox	Select from a lookup or fixed list.
listBox	Select one or more values from a list.
browse	Business record list.
grid	Cell-oriented tabular editing or review.
treeView	Hierarchical data.
tab	Multi-page layout.
groupBox	Visual grouping with optional caption.
image	Image display.
documentView	Formatted document or help display.
progressBar	Progress display.
menuBar	Window menu structure.
toolBar	Toolbar commands.
statusBar	Status messages.
timer	Timed event source.
reportView	Report preview or embedded report surface.

17.4 Data entry example

```
label NameLabel
   text "Customer name:"
   bounds 20, 20, 120, 20
end

entry NameEntry
   value Customer.Name
   bounds 150, 20, 240, 24
   maxLength 100
   required True
end

button SaveButton
   text "Save"
   bounds 20, 160, 90, 28
   on Click call SaveCustomer
end
```

17.5 Parent and child containment

```
panel AddressPanel
   bounds 20, 60, 460, 120

   label CityLabel
      text "City:"
      bounds 10, 10, 80, 20
   end

   entry CityEntry
      value Customer.City
      bounds 100, 10, 220, 24
   end
```

```
end
```

A child control belongs to its parent. Move, resize, hit testing, save, reload, layout overlay, and validation context follow the control tree.

18 Events and event handlers

Events connect user actions and runtime notifications to Safire procedures. An event handler should be short and should call a procedure or service when business logic is required.

18.1 Syntax

```
on Object.Event
   statements
end

on SaveButton.Click
   SaveCustomer()
end
```

18.2 Inline and call forms

```
button SaveButton
   text "Save"
   on Click call SaveCustomer
end

on CustomerBrowse.DoubleClick
   OpenCurrentCustomer()
end
```

18.3 Common events

Event	Meaning
Created	Control or window object has been created.
Opening	Window is about to open.
Opened	Window has opened.
Closing	Window is about to close.
Closed	Window has closed.
Click	User clicked a button or command control.
DoubleClick	User double-clicked a control or row.
Changed	Value changed.
ValueChanged	Value changed and should be handled.
SelectionChanged	Selection changed.
RowSelected	Browse or grid row changed.
RowAccepted	User accepted a row for action.
CellChanged	Grid cell value changed.
KeyDown	Key was pressed.
KeyUp	Key was released.
Validating	Value or form should be validated.
Command	Menu, toolbar, or command strip action selected.
Timer	Timer tick occurred.
BeforeRun	Report or service action is about to run.
AfterRun	Report or service action completed.

18.4 Event example

```
on CustomerBrowse.DoubleClick
```

```
    var Id is Integer
    Id = CustomerBrowse.CurrentRow.CustomerId
    OpenWindow(CustomerEditWindow, customerId = Id, mode = Change)
end
```

18.5 Event guidance

Good event handler style: on PostButton.Click call PostCurrentInvoice. Avoid placing all posting rules directly inside a button event. Put business rules in source-defined procedures, services, and validation rules.

19 Data binding

Data binding connects controls to fields, variables, query columns, object properties, or row selections. It is central to source-backed visual design.

19.1 Binding to fields

```
entry NameEntry
   value Customer.Name
end

checkbox ActiveCheck
   value Customer.IsActive
end
```

19.2 Binding to variables

```
window SearchWindow
   var SearchText is String(100)

   entry SearchEntry
      value SearchText
   end
end
```

19.3 Binding to a browse row

```
browse CustomerBrowse
   rows CustomerSearchQuery(SearchText = SearchText)
end

entry SelectedNameEntry
   value CustomerBrowse.CurrentRow.Name
end
```

19.4 Binding lifecycle

```
procedure LoadCustomer(CustomerId is Integer)
   Customer = CustomerGet(CustomerId)
   Bind(CustomerEditWindow, Customer)
end

procedure SaveCustomer()
   if not Validate(Customer) then
      ShowValidationMessages()
      return
   end

   Save(Customer)
end
```

19.5 Binding comments

A binding should make the data path clear. Avoid hidden data flow. A support person should be able to inspect the source and understand which field or object a control edits.

20 Browse and update patterns

A common business pattern is browse/update: a list of records with actions to add, change, view, delete, print, export, or drill into related data.

20.1 Browse declaration

```
browse CustomerBrowse for Customer
  source CustomerSearchQuery
  defaultSort Customer.Name
  column CustomerCode title "Code" width 90
  column Name title "Customer" width 260
  column Balance title "Balance" width 100 format "money"
  action New opens CustomerEditWindow mode Insert
  action Edit opens CustomerEditWindow mode Change
  action Delete calls DeleteCustomer
  action Print calls PrintCustomerList
end
```

20.2 Update window

```
window CustomerEditWindow
  title "Customer"
  dataSource Customer

  entry CodeEntry value Customer.CustomerCode
  entry NameEntry value Customer.Name
  entry EmailEntry value Customer.EmailAddress
  entry CreditLimitEntry value Customer.CreditLimit

  button SaveButton text "Save"
  button CancelButton text "Cancel"

  on SaveButton.Click
    SaveCustomer()
  end

  on CancelButton.Click
    CloseWindow()
  end
end
```

20.3 Generator rule

A generator may create starter browse/update source. After generation, the source remains developer-owned and must be readable, editable, buildable, and recoverable outside the visual designer.

21 Reports

Reports are first-class Safire artifacts. A report definition should be visible in source, editable by a report designer, and usable from runtime procedures.

21.1 Report declaration

```
report CustomerListReport
  title "Customer List"
  page A4 Portrait
  margins 12mm, 12mm, 12mm, 12mm
  dataSource CustomerSearchQuery
end
```

21.2 Bands

```
report CustomerListReport
  dataSource CustomerSearchQuery

  band PageHeader
    text "Customer List" at 10mm, 8mm fontSize 14 bold
    line at 10mm, 16mm to 190mm, 16mm
  end

  band Detail
    field CustomerCode at 10mm, 0mm width 25mm
    field Name at 40mm, 0mm width 80mm
    field Balance at 150mm, 0mm width 30mm align Right format "money"
  end

  band PageFooter
    text "Page " + PageNumber() at 170mm, 0mm width 20mm align Right
  end
end
```

21.3 Report elements

Element	Purpose
text	Static or calculated text.
field	Data-bound field.
expression	Calculated value.
line	Drawn line.
box	Rectangle or visual container.
image	Image from resource or data.
barcode	Barcode value.
checkbox	Boolean display.
total	Group or report total.

21.4 Groups and totals

```
group by Customer.Region
  header
    text Customer.Region at 10mm, 0mm bold
```

```
    end

    footer
        text "Region total:" at 120mm, 0mm
        total Balance sum at 150mm, 0mm format "money"
    end
end
```

21.5 Preview, print, and export

```
PreviewReport(CustomerListReport, SearchText = "")
PrintReport(CustomerListReport)
ExportReport(CustomerListReport,
    format = "PDF",
    fileName = "CustomerList.pdf")
```

22 Resources, configuration, and secrets

22.1 Resources

```
resource AppIcon = "resources/icons/app.ico"
resource Logo = "resources/images/logo.png"
resource HelpHome = "resources/help/index.html"
```

22.2 Configuration

```
config setting DatabasePath default "data/app.sdb"
config setting CompanyName default "Example Trading"

DbPath = Config("DatabasePath")
CompanyName = Config("CompanyName")
```

22.3 Secrets

```
Password = Secret("DbPassword")
CertificatePassword = Secret("CertificatePassword")
```

Secrets should not be stored as plain source values. They may be provided by the operating system, deployment environment, encrypted local store, or administrator configuration.

22.4 Resource comments

Resources should be declared so package tools can include them, diagnostics can report missing resources, and the application can be moved between machines without hidden dependencies.

23 Standard runtime object reference

Runtime objects give developers named, documented access to common runtime services.

Object	Common members	Purpose
FileObject	ReadText, WriteText, Exists, Copy	Read, write, test, and copy files.
PathObject	Normalize, Combine, FileName, DirectoryName	Work with safe and portable paths.
StringObject	Upper, Lower, Replace, Split, Trim	Common text operations.
DateTimeObject	ParseIso, FormatIso, Now	Date and time formatting and parsing.
JsonObject	Parse, Get, Set, Write	Structured data document handling.
XmlObject	Parse, Element, Text, Write	Structured markup document handling.
CsvObject	ReadRows, WriteRows	Delimited row data.
DiagnosticLog	Info, Warn, Error, Context	Structured diagnostics and support information.
HttpClient	Request, Response, Headers, Body	Client request and response handling.
HttpServer	Route, Request, Response, Status	Server route and response handling.
EmailClient	Message, To, Subject, Body, Send	Outbound message handling.
PdfDocument	NewPage, Text, Export	Document creation and export.
ReportDocument	LoadDefinition, BindRows, Preview, Export	Report runtime execution.
RuntimeError	Code, Message, Source, Line, Column	Structured error details.
ResultStatus	Ok, Failed, Value, Error	Success/failure result object.

23.1 File example

```
if FileObject.Exists("data/import.csv") then
   Text = FileObject.ReadText("data/import.csv")
end
```

23.2 Structured data example

```
CustomerJson = JsonObject.Parse(Text)
CustomerName = CustomerJson.Get("name")
CustomerJson.Set("status", "active")
OutputText = CustomerJson.Write()
```

23.3 Diagnostic example

```
DiagnosticLog.Info("Import started")
DiagnosticLog.Context("ImportFile", ImportFileName)
DiagnosticLog.Warn("Skipped blank row")
```

24 Communication and secure service objects

24.1 Client and server objects

Object	Purpose
HttpClient	Call a route or service endpoint and receive a response.
HttpRequest	Represents method, path, headers, body, and parameters.
HttpResponse	Represents status, headers, body, and content type.
HttpServer	Declares routes and executes request handlers.
HttpsClient	Client object with secure connection settings.
HttpsServer	Server object with secure connection settings.
EmailClient	Creates and sends messages through configured mail settings.
MessageObject	To, subject, body, attachments, and delivery settings.

24.2 Route declaration example

```
httpServer AppServer
   route GET "/health" call HealthCheck
   route GET "/customers" call ListCustomers
   route POST "/customers/save" call SaveCustomerRoute
end

function HealthCheck(Request is HttpRequest) returns HttpResponse
   return HttpResponse.Ok("OK")
end
```

24.3 Secure objects

Object	Purpose
TlsCertificate	Represents a certificate and its validation state.
TlsKey	Represents a private key reference.
TlsContext	Binds certificates, keys, protocol policy, and validation rules.
TlsServerOptions	Server-side secure connection settings.
TlsClientOptions	Client-side secure connection settings.
TlsSocket	Secure socket object.
CertificateStore	Certificate lookup and storage access.
CertificateRenewal	Renewal plan and renewal diagnostics.
MutualTlsPolicy	Client and server certificate trust policy.
TlsDiagnosticResult	Hostname, expiry, chain, protocol, and policy diagnostics.

24.4 Secure service example

```
cert = TlsCertificate.Load("certs/server.crt")
key = TlsKey.Load("certs/server.key")
context = TlsContext.Create(cert, key)

diag = cert.ValidateForHost("app.example.local")
if diag.Failed then
   ShowError(diag.Message)
   return
end
```

```
server = HttpsServer.Create(context)
server.Route("GET", "/health", HealthCheck)
server.Start()
```

25 Tasks, timers, and background work

Business applications often need long-running work, scheduled work, or timed refresh. Safire separates user interface responsiveness from background work.

25.1 Timer

```
timer RefreshTimer
  interval 30000
  enabled True
  on Tick call RefreshDashboard
end
```

25.2 Task

```
task ImportTask
  run ImportSupplierFile
  on Completed call ImportCompleted
  on Failed call ImportFailed
end
```

25.3 Guidance

Do not perform slow work directly inside a click event when it would freeze the user interface. Start a task, show progress, write diagnostics, and report completion to the user.

26 Build, run, package, and deploy

Safire projects are intended to be buildable outside the visual design environment. The same source that is edited visually should be checkable, buildable, runnable, and packageable through user-facing commands.

26.1 Typical workflow

```
safire check CustomerManager/project.sfproj
safire build CustomerManager/project.sfproj
safire run CustomerManager/project.sfproj
safire package CustomerManager/project.sfproj --configuration Release
```

26.2 Command meanings

Command	Purpose
safire check	Validate project structure, source files, references, bindings, resources, and common errors.
safire build	Compile and prepare the application output.
safire run	Run the application or project target.
safire package	Create a deployable application folder or installer-ready package.
safire diagnose	Collect project health, environment, configuration, and support information.
safire accept	Run the project-level developer acceptance workflow when available.

26.3 Deployment contents

A deployment normally contains application artifacts, runtime libraries, configuration files, database files or connection settings, report assets, resources, support tools, and a manifest describing the package.

26.4 Artifact types

Artifact type	Use
windows-exe	User-facing desktop application.
windows-service-exe	Service or server application that runs without an ordinary desktop window.
windows-runtime-dll	Runtime library or shared component.
web-app	Hosted web application package.
cli-tool	Command-line utility or maintenance tool.
report-pack	Report definitions and runtime assets.
portable-zip	Folder-based portable deployment.
installer-package	Installer-ready application package.

27 Diagnostics and support

Diagnostics are part of a maintainable business application. A support person should be able to see what project, version, settings, resources, database, report, runtime, route, and error information applies to the failing operation.

27.1 Runtime error object

```
try
   ImportCustomers(ImportFile)
catch ex is RuntimeError
   DiagnosticLog.Error(ex.Code + ": " + ex.Message)
   ShowError("Import failed. See diagnostics for details.")
end
```

27.2 Support bundle contents

Content	Purpose
Application version	Confirms exact deployed application.
Project manifest summary	Shows project identity and artifact type.
Configuration summary	Shows non-secret configuration values.
Runtime diagnostics	Shows errors, warnings, and context.
Database diagnostics	Shows database availability and schema state.
Report diagnostics	Shows report loading and export issues.
Resource diagnostics	Shows missing icons, images, strings, or documents.
Service diagnostics	Shows route, port, secure connection, and health state.

27.3 Diagnostic comments

Diagnostics must not expose secrets. Passwords, private keys, tokens, and protected values should be redacted or represented as present/missing status only.

28 Assistant-aware source practices

Safire source is designed to be readable by developers and by approved project assistance tools. Assistance should operate through reviewable changes, not hidden edits.

28.1 Good source practices

- Use clear object names.
- Keep event handlers short.
- Put business logic in procedures and classes.
- Declare data bindings explicitly.
- Keep generated output separate from source.
- Use comments for business rules that are not obvious from code.
- Define reusable validation rules in source.
- Keep reports, windows, dictionaries, and queries source-backed.

28.2 Reviewable change workflow

```
request
  -> collect project context
  -> propose source change
  -> review diff
  -> approve apply
  -> check/build/run
  -> keep diagnostic evidence
```

29 Language reference entries

This section lists common Safire language entries in a compact reference format.

29.1 application

Purpose: Defines application-level metadata and startup settings.

Developer usage: Use for title, version, main window, startup mode, default database, culture, and resources.

Typical example name: Application CustomerManager

```
application Name
   property value
end
```

29.2 module

Purpose: Groups related declarations into a named source module.

Developer usage: Use to organize shared procedures, classes, constants, and runtime object wrappers.

Typical example name: module CustomerRules

```
module Name
   declarations
end
```

29.3 use

Purpose: Makes a module or package visible to the current source file.

Developer usage: Use at the top of a source file for shared declarations.

Typical example name: use Safire.UI

```
use PackageOrModuleName
```

29.4 var

Purpose: Declares a mutable variable.

Developer usage: Use for values that change during execution.

Typical example name: var Quantity is Integer = 1

```
var Name is Type = value
```

29.5 const

Purpose: Declares a constant value.

Developer usage: Use for values that should not change at runtime.

Typical example name: const DefaultTaxRate is Decimal = 15.00

```
const Name is Type = value
```

29.6 type

Purpose: Declares a structured data type.

Developer usage: Use for records, summaries, transfer objects, and grouped business values.

Typical example name: type CustomerSummary

```
type Name
   fields
end
```

29.7 enum

Purpose: Declares a controlled list of named values.

Developer usage: Use for statuses, modes, categories, and fixed option sets.

Typical example name: enum InvoiceStatus

```
enum Name
   Value
end
```

29.8 class

Purpose: Declares a business object, service, rule set, or reusable component.

Developer usage: Use to group behavior and state.

Typical example name: class InvoiceService

```
class Name
   members
   methods
end
```

29.9 procedure

Purpose: Declares executable logic that does not return a value.

Developer usage: Use for commands and actions.

Typical example name: procedure SaveCustomer()

```
procedure Name(parameters)
   statements
end
```

29.10 function

Purpose: Declares executable logic that returns a value.

Developer usage: Use for calculations, checks, lookups, and services that return a result.

Typical example name: function CalculateVat(...) returns Money

```
function Name(parameters) returns Type
   statements
   return value
end
```

29.11 if

Purpose: Runs code conditionally.

Developer usage: Use for decision logic.

Typical example name: if Balance > CreditLimit then

```
if condition then
   statements
elseif condition then
   statements
else
   statements
end
```

29.12 case

Purpose: Chooses one branch from a set of known values.

Developer usage: Use when one value controls several alternatives.

Typical example name: case PaymentMethod

```
case expression
   when value
      statements
   else
      statements
end
```

29.13 loop

Purpose: Repeats statements.

Developer usage: Use for collections, counters, and while conditions.

Typical example name: loop for each Line in InvoiceLines

```
loop for each item in collection
   statements
end
```

29.14 try

Purpose: Handles runtime failures in a controlled way.

Developer usage: Use around operations that can fail and need user-facing recovery.

Typical example name: try PostInvoice(InvoiceId)

```
try
   statements
catch ex is Exception
   statements
end
```

29.15 validation

Purpose: Declares reusable business validation rules.

Developer usage: Use for form, import, service, and save validation.

Typical example name: validation CustomerRules for Customer

```
validation Name for Object
   rule RuleName
      when condition
      message "text"
   end
end
```

29.16 database

Purpose: Declares a database connection or store.

Developer usage: Use for data sources and deployment configuration.

Typical example name: database MainData

```
database Name
   property value
end
```

29.17 table

Purpose: Declares a database table or dictionary entity.

Developer usage: Use for persistent business data.

Typical example name: table Customer

```
table Name
   field declarations
   key declarations
end
```

29.18 field

Purpose: Declares a table field.

Developer usage: Use for typed columns, defaults, labels, required rules, and keys.

Typical example name: field Name is String(100) required

```
field Name is Type attributes
```

29.19 key

Purpose: Declares a lookup, primary, or uniqueness key.

Developer usage: Use for record identity and fast selection.

Typical example name: key CustomerCodeKey = CustomerCode unique

```
key Name = FieldList attributes
```

29.20 relation

Purpose: Declares a relationship between tables.

Developer usage: Use for parent-child and lookup relationships.

Typical example name: relation CustomerLink

```
relation Name
  from Field
  to Table.Field
end
```

29.21 query

Purpose: Declares reusable data selection.

Developer usage: Use for browse lists, reports, exports, and services.

Typical example name: query CustomerSearchQuery

```
query Name
  parameters
  from Table
  where condition
  order by Field
end
```

29.22 transaction

Purpose: Declares grouped work that must succeed or fail together.

Developer usage: Use for posting, imports, adjustments, and multi-record saves.

Typical example name: transaction PostInvoiceTransaction

```
transaction Name
```

```
  begin
    statements
  commit
  onError
    rollback
  end
end
```

29.23 window

Purpose: Declares a source-backed user interface window.

Developer usage: Use for forms, browses, dialogs, dashboards, and report windows.

Typical example name: window CustomerListWindow

```
window Name
  properties
  controls
  events
end
```

29.24 control

Purpose: Declares a generic control when a specific control keyword is not used.

Developer usage: Use for advanced or dynamically described controls.

Typical example name: control Search type entry

```
control Name type ControlType
  properties
end
```

29.25 on

Purpose: Declares an event handler.

Developer usage: Use for clicks, changes, row selection, open/close, commands, and timed events.

Typical example name: on SaveButton.Click

```
on Object.Event
  statements
end
```

29.26 bind

Purpose: Declares an explicit binding.

Developer usage: Use when a binding should be separate from the control declaration.

Typical example name: bind NameEntry.value to Customer.Name

```
bind Control.Property to DataPath
```

29.27 menuBar

Purpose: Declares a window menu structure.

Developer usage: Use for complete command access.

Typical example name: menuBar MainMenu

```
menuBar Name
   menu "Caption"
      command CommandName text "Caption"
   end
end
```

29.28 toolBar

Purpose: Declares a toolbar command strip.

Developer usage: Use for common commands.

Typical example name: toolBar MainToolbar

```
toolBar Name
   command CommandName text "Caption"
end
```

29.29 report

Purpose: Declares a report artifact.

Developer usage: Use for preview, print, and export output.

Typical example name: report InvoiceListReport

```
report Name
   properties
   bands
end
```

29.30 band

Purpose: Declares a report section.

Developer usage: Use for headers, footers, details, and groups.

Typical example name: band Detail

```
band Name
   report elements
end
```

29.31 resource

Purpose: Declares a named deployable asset.

Developer usage: Use for icons, images, strings, help, and document assets.

Typical example name: resource AppIcon = "resources/icons/app.ico"

```
resource Name = "path"
```

29.32 config

Purpose: Declares a configuration setting.

Developer usage: Use for deployment-specific values.

Typical example name: config setting DatabasePath default "data/app.sdb"

```
config setting Name default value
```

29.33 secret

Purpose: Reads a protected secret value.

Developer usage: Use for passwords, keys, tokens, and sensitive settings.

Typical example name: Password = Secret("DbPassword")

```
Secret("Name")
```

30 Complete example: customer manager

This example shows a compact customer manager application with application metadata, database source, table declaration, query, browse window, edit window, procedures, and validation.

```
application CustomerManager
    title "Customer Manager"
    version "1.0.0"
    mainWindow CustomerListWindow
    defaultDatabase MainData
end

database MainData
    provider "LocalStore"
    databaseName "data/customers.sdb"
end

table Customer
    field CustomerId is Integer primaryKey autoNumber
    field CustomerCode is String(20) required unique
    field Name is String(100) required
    field EmailAddress is String(255)
    field CreditLimit is Money default 0.00
    field Balance is Money default 0.00
    field IsActive is Boolean default True

    key CustomerCodeKey = CustomerCode unique
    key CustomerNameKey = Name
end

validation CustomerRules for Customer
    rule NameRequired
        when IsBlank(Customer.Name)
        message "Customer name is required."
    end

    rule CreditLimitPositive
        when Customer.CreditLimit < 0
        message "Credit limit may not be negative."
    end
end

query CustomerSearchQuery
    parameters
        SearchText is String(100) = ""
    end

    from Customer
    where Customer.Name contains SearchText
    order by Customer.Name
end

window CustomerListWindow
    title "Customers"
    bounds 40, 40, 900, 600
    startupPosition CenterScreen

    var SearchText is String(100) = ""
```

```
    entry SearchEntry
        text "Search"
        value SearchText
        bounds 10, 10, 300, 24
    end

    button SearchButton
        text "Search"
        bounds 320, 10, 80, 24
        on Click call RefreshCustomers
    end

    browse CustomerBrowse
        rows CustomerSearchQuery(SearchText = SearchText)
        bounds 10, 45, 860, 460
        column CustomerCode title "Code" width 90
        column Name title "Name" width 260
        column EmailAddress title "Email" width 240
        column Balance title "Balance" width 100 align Right format "money"
    end

    button NewButton text "New" bounds 10, 520, 80, 28
    button EditButton text "Edit" bounds 100, 520, 80, 28
    button CloseButton text "Close" bounds 790, 520, 80, 28

    on NewButton.Click
        OpenWindow(CustomerEditWindow, mode = Insert)
        Refresh(CustomerBrowse)
    end

    on EditButton.Click
        OpenCurrentCustomer()
    end

    on CustomerBrowse.DoubleClick
        OpenCurrentCustomer()
    end

    on CloseButton.Click
        CloseWindow()
    end
end

procedure OpenCurrentCustomer()
    if CustomerBrowse.CurrentRow.IsEmpty then
        ShowMessage("Select a customer first.")
        return
    end

    OpenWindow(CustomerEditWindow,
        mode = Change,
        customerId = CustomerBrowse.CurrentRow.CustomerId)

    Refresh(CustomerBrowse)
end

window CustomerEditWindow
    title "Customer"
```

```
    bounds 60, 60, 520, 300
    startupPosition CenterParent
    dataSource Customer

    entry CodeEntry text "Code" value Customer.CustomerCode
    entry NameEntry text "Name" value Customer.Name
    entry EmailEntry text "Email" value Customer.EmailAddress
    entry CreditLimitEntry text "Credit limit" value Customer.CreditLimit
    checkbox ActiveCheck text "Active" value Customer.IsActive

    button SaveButton text "Save"
    button CancelButton text "Cancel"

    on SaveButton.Click
        SaveCustomer()
    end

    on CancelButton.Click
        CloseWindow()
    end
end

procedure SaveCustomer()
    if not Validate(Customer) then
        ShowValidationMessages()
        return
    end

    Save(Customer)
    CloseWindow()
end

procedure RefreshCustomers()
    Refresh(CustomerBrowse)
end
```

31 Complete example: invoice report

This example shows how a table, query, report, and print procedure work together.

```
table Invoice
  field InvoiceId is Integer primaryKey autoNumber
  field InvoiceNumber is String(20) required unique
  field CustomerId is Integer required
  field InvoiceDate is Date required
  field TotalAmount is Money default 0.00

  key InvoiceNumberKey = InvoiceNumber unique
  key InvoiceDateKey = InvoiceDate
end

query InvoiceListQuery
  parameters
    FromDate is Date
    ToDate is Date
  end

  from Invoice
  where Invoice.InvoiceDate >= FromDate
    and Invoice.InvoiceDate <= ToDate
  order by Invoice.InvoiceDate, Invoice.InvoiceNumber
end

report InvoiceListReport
  title "Invoice List"
  page A4 Portrait
  margins 12mm, 12mm, 12mm, 12mm
  dataSource InvoiceListQuery

  band PageHeader
    text "Invoice List" at 10mm, 8mm fontSize 14 bold
    text "Date" at 10mm, 18mm bold
    text "Number" at 45mm, 18mm bold
    text "Customer" at 80mm, 18mm bold
    text "Total" at 160mm, 18mm width 30mm align Right bold
    line at 10mm, 25mm to 190mm, 25mm
  end

  band Detail
    field InvoiceDate at 10mm, 0mm width 30mm format "date"
    field InvoiceNumber at 45mm, 0mm width 30mm
    field Customer.Name at 80mm, 0mm width 75mm
    field TotalAmount at 160mm, 0mm width 30mm align Right format "money"
  end

  band ReportFooter
    text "Grand total:" at 120mm, 0mm width 35mm align Right bold
    total TotalAmount sum at 160mm, 0mm width 30mm align Right format "money" bold
  end
end

procedure PrintInvoiceList(FromDate is Date, ToDate is Date)
  PreviewReport(InvoiceListReport, FromDate = FromDate, ToDate = ToDate)
end
```

32 Appendix A: reserved words

The following words are reserved or should be treated as reserved by Safire tools. This list may grow as the language matures. Do not use reserved words as identifiers. application, and, as, band, boolean, break, browse, by, case, catch, class, config, const, constraint, continue, culture, database, date, datetime, decimal, default, do, else, elseif, end, enum, event, false, field, for, from, function, guid, if, import, in, include, integer, internal, key, layout, long, loop, map, menu, menuBar, money, new, not, null, on, or, page, panel, private, procedure, protected, public, query, real, ref, relation, report, resource, return, route, secret, string, table, task, text, then, this, throw, time, timer, transaction, true, try, type, use, validation, var, when, where, while, window, with

33 Appendix B: common runtime procedures and functions

Procedure or function	Purpose
OpenWindow(WindowName, ...)	Open a window.
CloseWindow()	Close the current window.
Refresh(Control)	Refresh a data display or control.
Display(Control)	Redisplay bound values.
Select(Control)	Give focus to a control.
ShowMessage(Text)	Show an information message.
ShowWarning(Text)	Show a warning message.
ShowError(Text)	Show an error message.
Validate(Object)	Run validation rules.
ValidateWithResult(Object)	Run validation and return a result object.
ShowValidationMessages()	Show collected validation messages.
Save(Object)	Save a record or object.
Delete(Object)	Delete a record or object.
ExecuteQuery(Query, ...)	Execute a query.
PreviewReport(Report, ...)	Preview a report.
PrintReport(Report, ...)	Print a report.
ExportReport(Report, Format, FileName)	Export a report.
Today()	Current date.
Now()	Current date and time.
Round(Value, Places)	Round a numeric value.
Trim(Text)	Remove leading and trailing spaces.
Upper(Text)	Uppercase text.
Lower(Text)	Lowercase text.
ToString(Value)	Convert a value to text.
Config(Name)	Read a configuration value.
Secret(Name)	Read a protected secret.
NewGuid()	Create a new unique identifier.

34 Appendix C: glossary

Term	Meaning
Application	The top-level Safire program artifact.
Artifact	A build, runtime, package, report, or deployment output.
Binding	A link between a control property and application data.
Browse	A list-oriented user interface pattern for selecting and managing records.
Class	A source object that groups state and behavior.
Configuration	Non-secret settings used by the application.
Control	A named user interface element.
Dictionary	Source-defined database, table, field, key, and relationship model.
Event	Runtime notification handled by Safire source code.
Form	A window or panel used for viewing or editing data.
Generated artifact	Reproducible output generated from source.
Package	Deployable application output.
Query	Reusable data selection.
Report	Printable or exportable source-defined output.
Resource	Named asset packaged with the application.
Runtime	Libraries and services needed to execute an application.
Secret	Protected value such as a password, key, or token.
Source-first	Rule that the source files remain the authoritative application truth.
Support bundle	Diagnostic package used to investigate build, runtime, license, configuration, or deployment issues.
Validation	Source-defined user-facing rule that checks values before work continues.
Window	Source-backed visual user interface artifact.

35 Closing note

Safire is intended to give business software developers a modern, source-backed development model: readable application source, visual design that writes back to source, clear business logic, strong data and report support, packaged applications, diagnostics, and assistance that remains under developer control.