Safire Language Reference

End User Guide and Developer Preview Reference

Version 1.0 Developer Preview

Safire Project - 2026-05-30

A practical source-language guide for building long-lived Windows business applications with source-backed UI, database dictionaries, queries, reports, build/deploy workflow, and controlled AI assistance.

This guide is product documentation. It intentionally describes Safire from the end-user and business developer perspective, not from the viewpoint of internal implementation history or external language influences.

Contents

1. Preface
2. Safire at a glance
3. Version 1 capability baseline
4. Project and source organization
5. Application declaration and startup
6. Source code format
7. Data declarations and types
8. Expressions and operators
9. Execution control
10. Procedures and functions
11. Classes and structured types
12. Error handling and validation
13. Dictionary and database declarations
14. Queries
15. Windows, forms, and business UI source
16. Control reference
17. Events and event handlers
18. Data binding
19. Browse and update patterns
20. Reports
21. Resources, configuration, and secrets
22. Import and external integration
23. AI-aware source and SafireIDE assistant pad
24. Build, run, package, and deploy
25. Licensing and product boundary
26. Language reference entries
27. Complete example: customer manager
28. Complete example: invoice report
29. Appendix A: reserved words
30. Appendix B: common runtime procedures
31. Appendix C: glossary

1. Preface

This document is an end-user language guide for Safire Version 1. It describes the public Safire language direction as it stands at the current Version 1 developer-preview baseline.

Safire is a Windows-first business application language and development ecosystem for experienced software developers who build, maintain, modernize, and extend long-lived business systems.

The guide is written as a practical language reference. It explains source structure, application declarations, variables, procedures, classes, database and dictionary declarations, windows, controls, events, data binding, browse/update patterns, reports, build and deployment concepts, licensing boundaries, and safe AI-assisted development.

Audience

This guide is for developers and technical decision makers who need to understand what Safire source code looks like, how business applications are represented, and how the SafireIDE productivity layer relates to the compiler, runtime, command-line tools, and deployable applications.

Version 1 note

Some examples describe the intended stable Safire source surface. In a developer-preview build, a specific feature may still be implemented through generated runtime plans, validation tools, or SafireIDE proof paths before it becomes a fully completed compiler/runtime feature. The language direction in this document is the source model that Safire is being aligned toward for Version 1.

Documentation conventions

Convention	Meaning
Keyword	A Safire language keyword or declaration name.
Name	A user-supplied identifier such as a window, report, table, field, procedure, or class name.
[item]	Optional item.
item ...	Item may repeat.
A | B	Choose one alternative.
End	Terminates a structure or block.
// comment	Source code comment.

2. Safire at a glance

Safire is designed around business software rather than short scripts or isolated code snippets. A Safire project is meant to be readable from source files, editable visually in SafireIDE, buildable from command-line tools, and deployable without requiring the end user to own the IDE.

Core product idea

- Readable business-oriented source code.
- Source-defined applications, windows, controls, events, reports, database dictionaries, queries, procedures, and classes.
- A compiler and runtime architecture that can check source and produce deployable artifacts.
- A licensed visual SafireIDE for productivity.
- A non-IDE source/build path for trust, maintenance, and anti-lock-in.
- An AI assistant pad that works with project context through reviewable, auditable changes.

Language personality

Safire source prefers named structures, explicit End termination, readable property assignment, and short event handlers. It is intended to feel calm and direct for business applications.

```
Application CustomerManager
    Title = "Customer Manager"
    Version = "1.0.0"
    MainWindow = WIN_CustomerList
End

Procedure Main()
    OpenWindow(WIN_CustomerList)
End
```

Source-first principle

Safire source files are the durable truth of the application. The IDE, designers, compiler, runtime previews, report editor, AI assistant, build tools, and deployment tools must read from and write back to the canonical source model.

> *Note: Generated output, previews, runtime plans, indexes, caches, logs, and temporary files are disposable. They must be reproducible from source.*

3. Version 1 capability baseline

The current Version 1 project direction provides a usable first product path rather than only a language experiment. The language and IDE are being aligned around practical business application delivery.

Current Version 1 areas

Area	Version 1 direction
SafireIDE	Project tree, source editor, visual designers, property/event/data panels, AI assistant pad, build/run/output loop, diagnostics, and support bundle workflow.
Source model	Source-first windows, reports, controls, events, dictionaries, queries, procedures, classes, project definitions, and deployment metadata.
Compiler and CLI	Check, build, run, and package workflow with regression protection for the command-line path.
Runtime	Runtime smoke-test application path, minimal database runtime, browse/update generation, report runtime, and end-user deployment packaging.
Reports	Source-backed report definitions with visual report design, bands, fields, grouping, totals, preview, print, and export direction.
AI assistant	Project-aware question answering, source context building, patch proposals, diff review, approved apply, backup, audit trail, and local-model integration path.
Licensing	Licensed professional IDE boundary, independent runtime/CLI use, and end-user deployment packages that do not include the licensed IDE.

Developer promise

A Safire developer should be able to open a project visually, inspect the underlying source, ask AI for help, review a proposed change, see a diff, approve the apply, build, run, and package the application without losing ownership of the project.

4. Project and source organization

A Safire project contains source files that describe business artifacts. SafireIDE presents these artifacts in a developer-friendly project tree, while the files remain readable and buildable outside the IDE.

Typical project layout

```
MyBusinessApp/
    project.sfproj
    app/
        Application.sf
    windows/
        WIN_CustomerList.sfwin
        WIN_CustomerEdit.sfwin
    reports/
        RPT_CustomerList.sfrpt
    dictionary/
        Customer.sfdict
        Invoice.sfdict
    queries/
        Q_CustomerSearch.sfquery
    classes/
        CustomerService.sf
        InvoiceService.sf
    procedures/
        GlobalProcedures.sf
    resources/
        icons/
        images/
    build/
        generated/
        output/
```

Source module types

Declaration	Purpose
Application	Defines the application-level source object.
Window	Defines a form or window and its controls.
Report	Defines a report, bands, printable elements, parameters, grouping, totals, and output behavior.
Table	Defines a database table or dictionary entity.
Query	Defines a reusable data selection for windows, browses, reports, services, or exports.
Class	Defines a business object, service, rule set, or reusable component.
Procedure	Defines executable code that performs an action.
Function	Defines executable code that returns a value.
Validation	Defines reusable business validation rules.

Resource	Defines deployable resources such as icons, images, and help files.

Generated output rule

Generated helpers are allowed, but they must not become hidden application truth. If a generated file is intentionally owned by the developer, it should be promoted into source. Otherwise, it belongs under a generated folder and can be rebuilt.

5. Application declaration and startup

The Application declaration describes the top-level business application. It is the natural place for title, version, publisher, startup window, default database, icon, culture, and deployment metadata.

Syntax

```
Application Name
    property = value
    ...
End
```

Example

```
Application SalesManager
    Title = "Sales Manager"
    Version = "1.0.0"
    Company = "Example Trading"
    MainWindow = WIN_Main
    DefaultDatabase = DB_Main
    Icon = Resource("icons/app.ico")
End
```

Common properties

Property	Description
Title	Display title of the application.
Version	Application version.
Company	Publisher or company name.
MainWindow	First window opened by the application.
DefaultDatabase	Default database connection or dictionary root.
Icon	Application icon resource.
Culture	Default language/culture setting.
LicenseMode	Optional application licensing/runtime mode where applicable.

Program entry point

A simple desktop business application can rely on MainWindow, or it can declare an explicit Main procedure.

```
Procedure Main()
    LoadConfiguration()
    OpenWindow(WIN_Main)
End
```

Service, console, maintenance, or import applications may run procedures instead of opening a window.

```
Procedure Main()
    LoadConfiguration()
    RunNightlyImport()
End
```

6. Source code format

Safire source is intended to be readable in a normal text editor and stable under visual designer round-trip.

Statements

A statement is normally written on one line. Prefer one logical action per line.

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

Blocks

Compound declarations and control structures are terminated with End.

```
If Balance > CreditLimit Then
    ShowWarning("Customer is over the credit limit.")
End
```

Comments

```
// Import orders that were captured offline.
ImportOfflineOrders()

/*
    Validation is separated from posting so that errors can be shown
    before any records are committed.
*/
ValidateBatch()
```

Identifiers

Identifiers name variables, procedures, fields, windows, reports, controls, classes, and other source objects. Use clear business names and avoid names that depend on temporary implementation details.

Object	Recommended example
Window	WIN_CustomerList
Report	RPT_Invoice
Query	Q_CustomerSearch
Button	BTN_Save
Label	LBL_CustomerName
Edit control	EDT_CustomerName
Browse/table control	TBL_Customers
Class	CustomerService
Procedure	PostInvoice

Field	CustomerName

7. Data declarations and types

Safire uses explicit declarations so that the compiler, designers, data binding, validation tools, and AI assistant can understand the application.

Variables

```
Var CustomerName is String(100)
Var Quantity is Integer = 1
Var UnitPrice is Money = 0.00
Var IsActive is Boolean = True
```

Constants

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

Common types

Type	Purpose	Example
Boolean	True/False values.	True
Integer	Standard whole number.	12345
Long	Larger whole number.	1234567890
Decimal	Fixed-precision decimal.	123.45
Money	Currency and business amounts.	99.95
Real	Floating point value.	3.14159
String(n)	Text up to length n.	String(100)
Text	Longer memo text.	Notes
Date	Calendar date.	Today()
Time	Time of day.	14:30:00
DateTime	Date and time.	Now()
Guid	Unique identifier.	NewGuid()
Blob	Binary content.	File data
Any	Controlled dynamic value.	Interop or generic routines

Strings

String literals use double quotes. Prefer clear string functions for business formatting.

```
CustomerName = "Acme Trading"
Message = "Invoice posted successfully."
```

Escape	Meaning
\n	New line.
\t	Tab.

\"	Double quote.
\\	Backslash.

Dates, times, and money

Date, time, Decimal, and Money values should be treated as typed values, not ordinary strings. Use Money or Decimal for financial calculations.

```
Var InvoiceDate is Date = Today()
Var PostedAt is DateTime = Now()
Var VatAmount is Money

VatAmount = Round(ExclusiveAmount * VatRate / 100, 2)
```

8. Expressions and operators

Expressions combine values, variables, fields, function calls, and operators.

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.

Arithmetic and comparison

```
LineTotal = Quantity * UnitPrice
InvoiceTotal = SubTotal + VatAmount

If Balance > CreditLimit Then
    ShowWarning("Credit limit exceeded.")
End
```

Operator	Meaning
+	Addition or string concatenation where supported.
-	Subtraction.
*	Multiplication.
/	Division.
%	Remainder or modulus where supported.
= or ==	Equal comparison in conditions.

<> or !=	Not equal.
< <= > >=	Less/greater comparisons.

Logical operators

```
If IsActive And Balance > 0 Then
    SendStatement(CustomerId)
End

If Not Customer.Exists(CustomerId) Then
    ShowError("Customer not found.")
End
```

Field qualification

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

9. Execution control

Execution control statements decide what code runs and how often.

If

```
If Balance > CreditLimit Then
    StatusText = "Over limit"
ElseIf Balance > 0 Then
    StatusText = "Open balance"
Else
    StatusText = "Clear"
End
```

Case

```
Case PaymentMethod
    When "CASH"
        PostCashPayment()
    When "CARD"
        PostCardPayment()
    When "EFT"
        PostBankPayment()
    Else
        ShowError("Unknown payment method.")
End
```

Loops

```
Loop For Each Line In InvoiceLines
    SubTotal += Line.LineTotal
End
```

```
Loop i From 1 To 10
    Trace("Line " + ToString(i))
End

Loop While Reader.HasMoreRows()
    ImportRow(Reader.CurrentRow)
    Reader.MoveNext()
End
```

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 Customer.Exists(CustomerId)
End
```

10. Procedures and functions

Procedures perform work. Functions perform work and return a value. Keep business logic in procedures, functions, and classes rather than placing too much logic directly in event handlers.

Procedure declaration

```
Procedure PostInvoice(InvoiceId is Integer)
    ValidateInvoice(InvoiceId)
    SaveLedgerEntries(InvoiceId)
    MarkInvoicePosted(InvoiceId)
End
```

Function declaration

```
Function CalculateVat(Amount is Money, Rate is Decimal) Returns Money
    Return Round(Amount * Rate / 100, 2)
End
```

Parameters

```
Procedure SendStatement(CustomerId is Integer,
                        EmailAddress is String(255))
    // Send customer statement.
End
```

Optional parameters

```
Procedure ShowMessage(Message is String,
                      Title is String = "Safire")
    MessageBox(Title, Message)
End
```

Reference parameters

```
Function TryGetCustomerName(CustomerId is Integer,
                            Ref Name is String) Returns Boolean
    If Not Customer.Exists(CustomerId) Then
        Return False
    End
    Name = Customer.Get(CustomerId).Name
    Return True
End
```

Note: Use reference parameters carefully. Prefer returning a result object when more than one output value is needed.

11. Classes and structured types

Safire classes and structured types let developers model business behavior and shared services.

Structured type

```
Type CustomerSummary
    CustomerId is Integer
    Name is String(100)
    Balance is Money
End
```

Class declaration

```
Class CustomerService
    Private Db is DatabaseConnection

    Public Procedure Open()
        Db = OpenDatabase(DB_Main)
    End

    Public Function FindName(CustomerId is Integer) Returns String
        Return Customer.Get(CustomerId).Name
    End
End
```

Visibility

Keyword	Meaning
Public	Available to other modules.
Private	Available only inside the class or module.
Protected	Available inside the class and derived classes.

Internal	Available inside the project or package.

Service-oriented business code

A common Safire pattern is to keep user interface event handlers short and move business rules into service classes.

```
Class InvoiceService
    Public Procedure Post(InvoiceId is Integer)
        ValidateBeforePost(InvoiceId)
        WriteLedgerEntries(InvoiceId)
        MarkInvoicePosted(InvoiceId)
    End

    Private Procedure ValidateBeforePost(InvoiceId is Integer)
        // Implementation hidden inside the class.
    End
End
```

12. Error handling and validation

Business applications need predictable error handling and clear validation messages. Safire separates technical errors from user-facing validation as much as possible.

Try and Catch

```
Try
    PostInvoice(InvoiceId)
Catch ex is Exception
    ShowError(ex.Message)
End
```

Throw

```
If InvoiceId <= 0 Then
    Throw New ValidationException("Invoice number is required.")
End
```

Validation block

Validation should be source-defined so that forms, reports, services, generators, and AI analysis can understand it.

```
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
```

User-facing validation

```
If Not Validate(Customer) Then
    ShowValidationMessages()
    Return
End
```

13. Dictionary and database declarations

Safire treats database-backed business systems as a primary use case. Database and dictionary concepts are visible to the developer rather than buried inside generic code.

Table declaration

```
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 K_Code = CustomerCode Unique
    Key K_Name = Name
End
```

Field properties

Property	Meaning
Required	Field must contain a value.
Default	Default value for new records.
PrimaryKey	Main record identifier.
AutoNumber	Runtime or database generates next value.
Unique	Duplicate values are not allowed.
DisplayName	User-facing field label.
Picture	Formatting or input mask.
Lookup	Links to a lookup table or list.
Validate	Attached validation rule.

Keys and relationships

```
Key K_Code = CustomerCode Unique
Key K_Name = Name
Key K_ActiveName = 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
```

Database connection

```
Database DB_Main
    Provider = "SQLite"
    DatabaseName = "data\sales.db"
End

Database DB_Accounts
    Provider = "ODBC"
    DataSource = "AccountsDSN"
    UserName = Config("DbUser")
    Password = Secret("DbPassword")
End
```

14. Queries

Queries define reusable data selections for windows, browses, reports, services, and exports. Use query declarations instead of duplicating SQL strings throughout the application.

Query declaration

```
Query Q_CustomerSearch
    Parameters
        SearchText is String(100) = ""
        ActiveOnly is Boolean = True
    End
    From Customer
    Where Customer.Name Contains SearchText
    Order By Customer.Name
End
```

SQL-backed query

```
Query Q_CustomerBalance
    Parameters
        MinimumBalance is Money
    End
    SQL
        SELECT CustomerId, Name, Balance
        FROM Customer
```

```
        WHERE Balance >= :MinimumBalance
        ORDER BY Name
    EndSQL
End
```

Query use

```
DataSet = ExecuteQuery(Q_CustomerBalance,
                       MinimumBalance = 1000.00)
```

Note: A query can be used by a browse, table control, report, export routine, background job, or business service.

15. Windows, forms, and business UI source

Windows are first-class Safire source artifacts. A visual designer can edit them, but the source remains authoritative.

Window declaration

```
Window WIN_CustomerList
    Title = "Customers"
    Size = 900, 600
    Center = True
    DataSource = Customer

    Table TBL_Customers
        At = 10, 10, 860, 480
        Source = Q_CustomerSearch
        Column CustomerCode Title "Code" Width 90
        Column Name Title "Customer" Width 260
        Column Balance Title "Balance" Width 100 Format "money"
    End

    Button BTN_New
        Caption = "New"
        At = 10, 510, 80, 28
    End

    Button BTN_Close
        Caption = "Close"
        At = 790, 510, 80, 28
    End
End
```

Common window properties

Property	Purpose
Title	Window title.
Size	Width and height.
Position	Initial screen position.
Center	Center on screen or parent.

Resizable	User may resize window.
MinSize	Minimum size.
Icon	Window icon.
DataSource	Default data source for binding.
Layout	Layout manager or sizer declaration.

Coordinates and layout

Safire supports coordinate-based placement for RAD familiarity and layout-based design for maintainable resizing.

```
Button BTN_Save
    Caption = "Save"
    At = 20, 120, 90, 28
End

Layout Vertical
    Margin = 10
    Gap = 8
    Add TBL_Customers Expand = True
    Add ButtonPanel Height = 36
End
```

Note: Safire hides low-level UI complexity behind a business-oriented source model. The developer declares windows, controls, properties, events, and bindings in source.

16. Control reference

Controls are declared inside a window, page, group, panel, toolbar, or report band. The SafireIDE designer should expose table-level properties separately from column-level properties so browse and table controls remain easy to manage.

Basic controls

```
Label LBL_Name
    Caption = "Customer name:"
    At = 20, 20, 120, 20
End

Edit EDT_Name
    Binding = Customer.Name
    At = 150, 20, 240, 24
    MaxLength = 100
End

Button BTN_Save
    Caption = "Save"
    At = 20, 160, 90, 28
End

CheckBox CHK_Active
```

```
    Caption = "Active customer"
    Binding = Customer.IsActive
End
```

Lookup and selection controls

```
ComboBox CBO_Terms
    Binding = Customer.PaymentTermsId
    Lookup = PaymentTerms.PaymentTermsId,
             PaymentTerms.Description
End

ListBox LST_Status
    Items = "New", "Active", "On Hold", "Closed"
    Binding = Customer.Status
End
```

Table and browse

```
Table TBL_Customers
    Source = Q_CustomerSearch
    Column CustomerCode Title "Code" Width 90 Sortable = True
    Column Name Title "Name" Width 260 Searchable = True
    Column Balance Title "Balance" Width 100 Align = Right
End
```

Common control catalog

Control	Typical use
Label	Static text.
Edit	Text, numeric, date, or bound entry.
Button	Command or action.
CheckBox	Boolean selection.
RadioButton	Exclusive option selection.
ComboBox	Select from lookup or list.
ListBox	Select from list.
Table	Rows and columns / browse.
TreeView	Hierarchical data.
Tab	Multi-page layout.
Panel	Container.
GroupBox	Visual grouping.
Image	Display image.
HtmlView	HTML content, help, or formatted preview.
ProgressBar	Progress display.
MenuBar	Window menu.
ToolBar	Toolbar actions.

StatusBar	Status display.
Timer	Timed events.

17. Events and event handlers

Events connect user actions and runtime notifications to Safire code. Event handlers should be short and should call procedures or services for business logic.

Event handler syntax

```
On Control.Event
    statements
End

On BTN_Save.Click
    SaveCustomer()
End

On Window.Opened
    LoadCustomerList()
End
```

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 control.
DoubleClick	User double-clicked.
Changed	Value changed.
ValueChanged	Value changed and should be handled.
SelectionChanged	Selection changed.
RowSelected	Browse/table row changed.
KeyDown	Key pressed.
KeyUp	Key released.
Validating	Value should be validated.
Drop	Drag-and-drop data arrived.
Timer	Timer tick.

Event example

```
On TBL_Customers.DoubleClick
    Var Id is Integer
    Id = TBL_Customers.CurrentRow.CustomerId
    OpenWindow(WIN_CustomerEdit, CustomerId = Id)
End
```

Note: Good event handler: On BTN_Post.Click -> PostCurrentInvoice(). Avoid placing all invoice posting rules directly inside the button click event.

18. Data binding

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

Binding to fields

```
Edit EDT_Name
    Binding = Customer.Name
End

CheckBox CHK_Active
    Binding = Customer.IsActive
End
```

Binding to variables

```
Window WIN_Search
    Var SearchText is String(100)

    Edit EDT_Search
        Binding = SearchText
    End
End
```

Binding to a table row

```
Table TBL_Customers
    Source = Q_CustomerSearch
End

Edit EDT_Name
    Binding = TBL_Customers.CurrentRow.Name
End
```

Binding lifecycle

A typical data-bound edit form follows this lifecycle: open form, load record, bind fields, edit, validate, save, refresh browse.

```
Procedure LoadCustomer(CustomerId is Integer)
    Customer = Customer.Get(CustomerId)
    Bind(WIN_CustomerEdit, Customer)
End
```

```
Procedure SaveCustomer()
    If Not Validate(Customer) Then
        ShowValidationMessages()
        Return
    End
    Customer.Save(Customer)
End
```

19. Browse and update patterns

Business applications often use a browse/update pattern: a list of records with actions to add, change, view, delete, print, export, or drill into related data.

Browse declaration

```
Browse BRW_Customers For Customer
    Source = Q_CustomerSearch
    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 WIN_CustomerEdit Mode Insert
    Action Edit Opens WIN_CustomerEdit Mode Change
    Action Delete Calls DeleteCustomer
    Action Print Calls PrintCustomerList
End
```

Update window

```
Window WIN_CustomerEdit
    Title = "Customer"
    DataSource = Customer

    Edit EDT_Code Binding = Customer.CustomerCode
    Edit EDT_Name Binding = Customer.Name
    Edit EDT_Email Binding = Customer.EmailAddress
    Edit EDT_CreditLimit Binding = Customer.CreditLimit

    Button BTN_Save Caption = "Save"
    Button BTN_Cancel Caption = "Cancel"

    On BTN_Save.Click
        SaveCustomer()
    End

    On BTN_Cancel.Click
        CloseWindow()
    End
```

```
End
```

Generator rule

A Safire generator may create starter browse/update source. After generation, the resulting source should remain readable and maintainable. The developer should be able to inspect, edit, build, and recover it outside the designer.

20. Reports

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

Report declaration

```
Report RPT_CustomerList
    Title = "Customer List"
    Page = A4 Portrait
    Margins = 12mm, 12mm, 12mm, 12mm
    DataSource = Q_CustomerSearch
End
```

Bands

```
Report RPT_CustomerList
    DataSource = Q_CustomerSearch

    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
```

Elements, groups, and totals

Element	Purpose
Text	Static label or calculated text.
Field	Bound data field.
Expression	Calculated value.
Line	Drawn line.
Box	Rectangle.
Image	Image from resource or data.

Barcode	Barcode value.
CheckBox	Boolean display.
Total	Group or report total.

```
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
```

Preview, print, and export

```
PreviewReport(RPT_CustomerList, SearchText = "")
PrintReport(RPT_CustomerList)
ExportReport(RPT_CustomerList,
             Format = "PDF",
             FileName = "CustomerList.pdf")
```

Note: Report designer changes should round-trip to source and be visible as source diffs.

21. Resources, configuration, and secrets

Safire applications need predictable access to files, resources, configuration, and protected secrets.

Resources

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

Configuration

```
Config Setting DatabasePath Default "data\app.db"
Config Setting CompanyName Default "Example Trading"

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

Secrets

Secrets should not be stored as plain source values.

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

Secrets may be provided by the operating system, deployment environment, encrypted local store, or administrator configuration.

22. Import and external integration

Safire is Windows-first and supports explicit external/native integration where required. Interop should be reviewable and should not be used to bypass normal business runtime services without a clear reason.

Import declaration

```
Import Native "kernel32.dll"
    Function GetTickCount() Returns Integer Alias "GetTickCount"
End
```

Calling imported functions

```
Ticks = GetTickCount()
```

Appropriate uses

- Operating system integration.
- Device SDK integration.
- Existing business DLLs.
- Migration support.
- Specialist libraries not provided by the Safire runtime.

Note: Prefer Safire database, UI, report, file, configuration, and runtime services for normal application work.

23. AI-aware source and SafireIDE assistant pad

Safire source is designed to be readable by developers and analyzable by the SafireIDE AI assistant pad. The assistant is project-aware, not an uncontrolled code generator.

AI-readable source practices

- Use clear object names.
- Keep event handlers short.
- Place business logic in procedures and classes.
- Declare data binding explicitly.
- Keep generated files 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.

AI patch workflow

The preferred AI workflow protects developer control:

```
Question or request
    -> project/source context build
    -> AI proposal
    -> human review
    -> source diff
    -> approved apply
    -> backup and audit trail
    -> build/test proof
```

Local model and hosted model boundary

Where a local or hosted model backend is used, Safire-controlled tooling should remain the authority over project files, patch application, backups, and build/test execution. The model supplies reasoning and proposals; SafireIDE controls the project.

24. Build, run, package, and deploy

Safire has a real compiler/runtime direction. Source is parsed and checked, compiler output can target runtime metadata and executable packaging, and deployments should not require the end user to own SafireIDE.

Build concepts

Concept	Meaning
Source	Developer-owned files.
Compiler	Parses and checks source.
Runtime metadata	Information used by forms, reports, binding, and runtime services.
Runtime	Libraries and services needed to execute an application.
CLI tools	Command-line check/build/run/package path.
SafireIDE	Licensed visual productivity environment.
EXE/DLL package	Deployable application artifact where required.
Runtime package	Runtime libraries, drivers, report/PDF support, and configuration.

Typical command-line workflow

```
safire check MyProject.sfproj
safire build MyProject.sfproj
safire run MyProject.sfproj
safire package MyProject.sfproj --configuration Release
```

Deployment contents

- Compiled EXE and/or DLL artifacts where required.
- Safire runtime components where needed.
- Database drivers.
- Report, print, PDF, and export runtime support.
- Configuration files.
- Runtime licensing keys where applicable.
- Support bundle tools.
- Installer or update assets.

Note: End users should not need SafireIDE to run a deployed Safire application.

25. Licensing and product boundary

Safire separates the licensed professional IDE from the developer-owned source and the runtime path. This is an important trust boundary.

Licensed SafireIDE

SafireIDE is the professional productivity product. It may include visual designers, project explorer, source editor, property/event/data panels, report designer, AI assistant pad, review history, templates, wizards, build/run/deploy workflow, support bundle creation, and other productivity features.

Non-IDE path

The non-IDE path exists to preserve trust. A developer should be able to inspect source, edit with a text editor, run command-line checks/builds, and maintain a project even without relying on hidden IDE state.

Runtime and end-user deployment

End-user deployments should contain only the application and runtime components needed to run the application. The licensed SafireIDE itself is not part of ordinary end-user deployment.

Item	Licensed IDE required?

Designing visually in SafireIDE	Yes.
Using full AI assistant pad in SafireIDE	Yes, depending on edition.
Editing source in a text editor	No.
Running command-line build tools where licensed/available	No IDE session required.
Running a deployed end-user application	No.
Including SafireIDE in an end-user package	No, and normally not allowed.

Note: Licensing details may vary by product edition. This guide describes the product boundary, not a legal license agreement.

26. Language reference entries

Application

```
Application Name
    property = value
End
```

Defines application-level metadata and startup settings.

Window

```
Window Name
    property = value
    controls
    events
End
```

Defines a source-backed form or window.

Report

```
Report Name
    property = value
    bands
End
```

Defines a report source artifact.

Procedure

```
Procedure Name(parameters)
    statements
End
```

Defines executable code that does not return a value.

Function

```
Function Name(parameters) Returns Type
    statements
    Return value
End
```

Defines executable code that returns a value.

Class

```
Class Name
    members
    methods
End
```

Defines a business object, service, rule set, or reusable component.

Table

```
Table Name
    Field declarations
    Key declarations
    Relation declarations
End
```

Defines a database table or dictionary source object.

Query

```
Query Name
    query definition
End
```

Defines reusable data selection.

On

```
On Object.Event
    statements
End
```

Declares an event handler.

27. Complete example: customer manager

This example shows a small business application using an application declaration, database, table, query, browse window, edit window, procedures, and validation.

```
Application CustomerManager
    Title = "Customer Manager"
    Version = "1.0.0"
    MainWindow = WIN_CustomerList
    DefaultDatabase = DB_Main
End

Database DB_Main
    Provider = "SQLite"
    DatabaseName = "data\customers.db"
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 K_Code = CustomerCode Unique
    Key K_Name = Name
End

Query Q_CustomerSearch
    Parameters
        SearchText is String(100) = ""
    End
    From Customer
    Where Customer.Name Contains SearchText
    Order By Customer.Name
End
Window WIN_CustomerList
    Title = "Customers"
    Size = 900, 600
    Center = True

    Var SearchText is String(100) = ""

    Edit EDT_Search
        Caption = "Search"
        Binding = SearchText
        At = 10, 10, 300, 24
    End

    Button BTN_Search Caption = "Search" At = 320, 10, 80, 24

    Table TBL_Customers
        Source = Q_CustomerSearch(SearchText = SearchText)
        At = 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 BTN_New Caption = "New" At = 10, 520, 80, 28
    Button BTN_Edit Caption = "Edit" At = 100, 520, 80, 28
    Button BTN_Close Caption = "Close" At = 790, 520, 80, 28

    On BTN_Search.Click
```

```
        Refresh(TBL_Customers)
    End

    On BTN_New.Click
        OpenWindow(WIN_CustomerEdit, Mode = Insert)
        Refresh(TBL_Customers)
    End

    On BTN_Edit.Click
        OpenCurrentCustomer()
    End

    On TBL_Customers.DoubleClick
        OpenCurrentCustomer()
    End

    On BTN_Close.Click
        CloseWindow()
    End
End
Procedure OpenCurrentCustomer()
    If TBL_Customers.CurrentRow.IsEmpty Then
        ShowMessage("Select a customer first.")
        Return
    End

    OpenWindow(WIN_CustomerEdit,
               Mode = Change,
               CustomerId = TBL_Customers.CurrentRow.CustomerId)
    Refresh(TBL_Customers)
End

Window WIN_CustomerEdit
    Title = "Customer"
    Size = 520, 300
    Center = True
    DataSource = Customer

    Edit EDT_Code Caption = "Code" Binding = Customer.CustomerCode
    Edit EDT_Name Caption = "Name" Binding = Customer.Name
    Edit EDT_Email Caption = "Email" Binding = Customer.EmailAddress
    Edit EDT_Credit Caption = "Credit limit" Binding = Customer.CreditLimit
    CheckBox CHK_Active Caption = "Active" Binding = Customer.IsActive

    Button BTN_Save Caption = "Save"
    Button BTN_Cancel Caption = "Cancel"
```

```
    On BTN_Save.Click
        If Not Validate(Customer) Then
            ShowValidationMessages()
            Return
        End
        Save(Customer)
        CloseWindow()
    End

    On BTN_Cancel.Click
        CloseWindow()
    End
End
```

28. Complete example: invoice report

This example shows how business data, query output, and a report definition 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 K_Number = InvoiceNumber Unique
    Key K_Date = InvoiceDate
End

Query Q_InvoiceList
    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 RPT_InvoiceList
    Title = "Invoice List"
    Page = A4 Portrait
    Margins = 12mm, 12mm, 12mm, 12mm
    DataSource = Q_InvoiceList

    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(RPT_InvoiceList, FromDate = FromDate, ToDate = ToDate)
End
```

29. 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.

```
Application And As Boolean Break Browse By Case Catch Class Const Continue
Database Date DateTime Decimal Default Do Else ElseIf End Enum Event False
Field For From Function Global Guid If Import In Include Integer Internal Key
Long Loop Money New Not On Or Private Procedure Protected Public Query Real
Ref Relation Report Resource Return String Table Text Then This Throw Time
True Try Type Validation Var When Where While Window With
```

Do not use reserved words as identifiers.

30. Appendix B: common runtime procedures

Procedure or function	Purpose
OpenWindow(WindowName, ...)	Open a window.
CloseWindow()	Close current window.
Refresh(Control)	Refresh data display.
Display([Control])	Redisplay bound values.
Select(Control)	Give focus to a control.
ShowMessage(Text)	Show information message.

ShowWarning(Text)	Show warning message.
ShowError(Text)	Show error message.
Validate(Object)	Run validation rules.
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/time.
Round(Value, Places)	Round numeric value.
Config(Name)	Read configuration value.
Secret(Name)	Read a protected secret.

31. Appendix C: glossary

Term	Meaning
Application	The top-level Safire program artifact.
Browse	A list-oriented user interface pattern for selecting and managing records.
Dictionary	Source-defined database/table/field model.
Event	Runtime notification handled by source code.
Form/window	Visual user interface artifact.
Generated artifact	Reproducible output generated from source.
IDE	Safire visual development environment.
Query	Reusable data selection.
Report	Printable or exportable source-defined output.
Runtime	Libraries and services needed to execute an application.
Source-first	Rule that the source files remain the authoritative application truth.
Support bundle	Diagnostic package used to help investigate build, license, runtime, or deployment issues.

Closing note

Safire is intended to give business software developers a familiar but modern development model: source-backed visual design, clear business logic, strong database and report support, buildable artifacts, deployable applications, and AI assistance that remains under developer control.