SAFIRE

Language Reference and End User Guide

Post-M651 Developer Preview Baseline

For experienced business-application developers evaluating SafireIDE and the Safire language

Developer Preview Reference - prepared from the current Safire project progress after M651.

Document type	Language reference, end user guide, and SafireIDE orientation
Audience	Mature business-software developers, consultants, and technical evaluators
Product state	Post-M651 developer preview - source-first IDE and designer path
Primary focus	Windows business applications, forms, dictionaries, queries, reports, build/run, and controlled AI assistance
Important note	Examples describe the intended stable source surface. Some features may still be implemented through runtime plans, IDE proof paths, generators, or developer-preview tooling.

Public positioning note: *This guide describes Safire from the user and business-developer viewpoint. It intentionally avoids internal implementation history and focuses on the usable product model, source ownership, and licensing boundary.*

Contents

1. Preface and document scope

This document is an end-user language reference and practical guide for the Safire developer-preview baseline after M651. It describes how Safire source is expected to represent business applications, how SafireIDE works with that source, and how the compiler, runtime, designers, reports, dictionaries, and controlled AI assistance fit together.

The guide is written for experienced business-application developers. It assumes the reader already understands forms, data entry, reports, table maintenance, deployment responsibility, and the need to keep customer systems stable over many years.

The document is not a promise that every keyword shown here is complete in every preview build. It is a reference for the current product direction and the source model that the IDE, runtime plans, designers, compiler path, and generated business workflows are being aligned toward.

1.1 Reading conventions

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

1.2 Public product stance

- Safire is a Windows-first business application development language and ecosystem.
- SafireIDE is the licensed professional productivity environment.
- Safire source files remain the durable truth of the application.
- The compiler and command-line path must remain useful enough for non-IDE maintenance and build confidence.
- AI assistance must be reviewable, reversible, and auditable.
- Version 1 is focused on practical business systems: forms, dictionaries, queries, reports, validation, and deployment.

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.

2.1 Core product idea

- Readable business-oriented source code.
- Source-defined applications, windows, controls, events, dictionaries, queries, reports, 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.

- A project-aware AI Assistant Pad that works through reviewable, auditable changes.

2.2 Simple application example

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

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

The example shows the intended personality of the language: named blocks, readable properties, and explicit business actions. Safire is meant to express what the business application is, not to force the developer to manually describe every low-level UI plumbing detail.

3. Post-M651 product baseline

M651 is important because it confirms the professional designer path is moving away from a demo canvas and toward actual source-backed form editing. After the M651 hotfix parser work, the designer source loader can open the actual active .sfform surface, clear starter controls, and leave empty source forms blank rather than displaying hardcoded starter content.

3.1 Baseline capabilities after M651

Area	Post-M651 meaning for the developer
SafireIDE shell	Ribbon/menu command surface, project explorer, tabbed designer/source area, properties/events/AI panels, and output panes are in place as the professional workbench direction.
Source-first model	Forms, reports, dictionaries, queries, procedures, classes, and project files are treated as source-owned artifacts rather than hidden designer-only state.
Professional form designer	The designer can be launched/integrated from the IDE path and is being aligned to open actual .sfform files and write back to source.
M651 parser correction	The designer no longer depends on starter controls as the source of truth. Actual source form metadata is loaded; empty forms remain blank.
Controls and events	Standard UI controls and event catalog direction exist, with event stubs and runtime dispatch already proven through earlier milestones.
Data model	Runtime record buffer, local data driver, binding, browse/list population, and CRUD patterns have

	reached early proof paths.
Reports	Report source, report editor, preview/print/PDF direction, validation, templates, and runtime integration have reached a first usable product path.
AI workflow	Local/project-aware AI direction includes context, proposals, diffs, approved apply, backup, audit history, build/test proof, and restore direction.
Build/run/deploy	The build and run active project path has been proven; copied release and packaging work have improved the move toward deployable application output.
Licensing	The licensed IDE is the productivity product, while source and build paths remain important for developer trust.

3.2 What M651 changes in user language

- A form source file is no longer just an abstract file that the designer may replace with a starter layout.
- The designer is expected to reflect the active .sfform source surface.
- Blank forms are legitimate source artifacts and should open blank.
- The user-owned source model is reinforced: the designer surface follows source rather than hiding source.
- The business developer can rely on the direction that visual editing and text editing are converging on the same canonical model.

3.3 Practical maturity statement

After M651, Safire should be described as a developer-preview business application system with a serious source-first IDE direction. It is not yet a finished commercial V1, but it now has enough product structure to document application source, form source, report source, dictionary source, controlled AI assistance, build/run behavior, and licensing boundaries in a coherent end-user guide.

4. SafireIDE end user workflow

SafireIDE is the main productivity environment for professional Safire development. It should present a calm business application workbench rather than exposing confusing internal artifacts first.

4.1 Typical daily workflow

1. Open a Safire project file.
2. Review the project tree: windows, reports, dictionaries, queries, procedures, classes, and resources.
3. Open a window/form or report in the designer.
4. Edit controls, properties, bindings, events, validation, and layout.
5. Preview proposed source write-back where appropriate.
6. Approve the write-back and let Safire create backup/audit evidence.
7. Build the active project.
8. Run the generated application executable.

9. Use the AI Assistant Pad for explanation, proposals, and diagnostics where helpful.
10. Package or stage the application for deployment when ready.

4.2 Recommended project explorer organization

Node	What belongs here
Project	Project manifest, build settings, package settings, metadata visible to the developer.
Windows / Forms	Source-backed .sfform or .sfwin files representing data entry screens, browse windows, and application dialogs.
Reports	Source-backed reports with bands, fields, grouping, totals, page setup, and output definitions.
Dictionaries / Analysis	Tables, fields, keys, relationships, validation, and lookup metadata.
Queries	Reusable data selections for browses, lookups, reports, exports, and services.
Procedures	Event handlers and procedural business actions.
Classes	Business services, rules, shared logic, and structured behavior.
Resources	Icons, images, help files, templates, and deployment resources.
Generated Files	Clearly separated derived output that can be rebuilt from source.

4.3 What the designer should not do

- It should not make hidden designer metadata the only source of a window.
- It should not silently discard developer-owned source.
- It should not replace an empty source form with a hardcoded sample surface.
- It should not apply AI-generated changes without review, diff, backup, and audit.
- It should not force end users to install SafireIDE to run deployed applications.

5. Project and source organization

A Safire project folder should separate source, generated output, build output, resources, and deployment packages. The layout below is a recommended user-facing structure for business projects.

```
CustomerManager/
  project.sfproj
  app/
    Application.sf
  windows/
    CustomerCards.sfform
    CustomerBrowse.sfform
    MainApplicationWindow.sfform
  reports/
    CustomerList.sfreport
    InvoiceSummary.sfreport
  dictionaries/
```

```
    customer.sdict
    invoice.sdict
  tables/
    Customer.sftable
    Invoice.sftable
    InvoiceLine.sftable
  queries/
    CustomerBrowseView.sfquery
    OrdersByCustomer.sfquery
  procedures/
    CustomerActions.sf
    ReportActions.sf
  classes/
    CustomerService.sf
  resources/
    icons/
    images/
    help/
  build/
    generated/
    output/
  release/
    package output here
```

5.1 Common source file types

Extension	Purpose
.sfproj / .safproj / .sproj	Project manifest and project-level build/deploy metadata. Different proof paths may still use more than one legacy extension while the project model stabilizes.
.sf	General Safire source module for procedures, functions, classes, and application declarations.
.sfform / .sfwin	Source-backed window/form declaration. .sfform is common in the current designer proof path.
.sfreport / .sfrpt	Source-backed report declaration.
.sdict / .sfdict	Dictionary or analysis source containing table, field, relationship, key, and validation metadata.
.sftable	Source-defined table metadata used by runtime/compiler/data tooling.
.sfquery	Reusable query declaration.
.sfstyle	Optional style/theme declarations owned by the developer when used.

5.2 Source ownership rule

Any project information that determines application behavior must be recoverable from developer-owned source. Generated previews, runtime plans, indexes, caches, proof logs, and intermediate code are disposable unless explicitly promoted into source.

6. Source code format

Safire source is intended to be readable in a normal text editor and stable under visual designer round-trip. The syntax style favors named blocks, property assignment, and explicit End terminators.

6.1 Statements

```
CustomerName = FirstName + " " + Surname
Balance += InvoiceTotal
Refresh(TBL_Customers)
SetText(EDT_Status, "Ready")
```

6.2 Blocks

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

For Each Customer In CustomerQueue
   ExportCustomer(Customer)
End
```

6.3 Comments

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

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

6.4 Style rules for business code

- Use clear business names.
- Keep event handlers short.
- Move business rules into procedures, functions, or classes.
- Prefer source comments that explain business intent rather than obvious syntax.
- Keep generated code separated from owned source.
- Use predictable prefixes for UI controls and business artifacts.

7. Identifiers, names, and reserved words

Identifiers name source artifacts such as windows, tables, fields, controls, procedures, functions, classes, queries, reports, variables, and constants. Consistent names help the developer, designer, compiler, runtime plan generator, and AI assistant understand the application.

7.1 Recommended naming patterns

Object type	Pattern	Example
Application	BusinessName	CustomerManager
Window/form	WIN_BusinessPurpose or	WIN_CustomerList,

	readable form name	CustomerCards
Report	RPT_BusinessPurpose	RPT_InvoiceSummary
Query	Q_BusinessPurpose	Q_CustomerSearch
Button	BTN_Action	BTN_Save
Edit control	EDT_FieldName or txtTableField	EDT_CustomerName, txtCustomerEmail
Check box	CHK_FieldName	CHK_Active
Browse/table control	TBL_ObjectPlural	TBL_Customers
Class/service	BusinessNameService	CustomerService
Procedure	VerbBusinessObject	SaveCustomer

7.2 Reserved words

The exact reserved list will harden with the compiler. Avoid using the following words as user identifiers: Application, Window, Form, Report, Table, Field, Key, Query, View, Queue, Class, Procedure, Function, Event, On, End, If, Then, Else, For, Each, While, Return, Try, Catch, Finally, Throw, True, False, Null, New, Public, Private, Import, Export, Validation, Transaction, Begin, Commit, Rollback.

8. Data declarations and value types

Safire data types are designed for business applications. The type system should support compiler checking, runtime values, database mapping, form binding, report formatting, validation, and AI analysis.

8.1 Core value types

Type	Purpose	Example
String	Names, descriptions, codes, addresses, references.	CustomerName String
Text	Longer multi-line content or notes.	Notes Text
Integer	Whole-number identifiers and counts.	Quantity Integer
Long	Large integer values.	AuditNumber Long
Decimal	Business numeric values requiring precision.	CreditLimit Decimal
Currency	Money values.	InvoiceTotal Currency
Boolean	True/False values.	Active Boolean
Date	Calendar date.	InvoiceDate Date
Time	Time of day.	StartTime Time
DateTime	Timestamp values.	CreatedAt DateTime
Blob	Binary content.	DocumentImage Blob
Guid	Globally unique identifier.	ExternalId Guid
Nullable<T>	Optional business value.	Nullable<Date> PaidDate
Row / Record	Runtime table row or record buffer.	CustomerRow Row
Object / Map	Named dynamic values for integration or metadata.	Options Map

8.2 Variable declaration examples

```
CustomerName String = "Sample Customer"
CreditLimit Decimal = 25000.00
Active Boolean = True
CreatedAt DateTime = Now()
Balance Currency = 0.00

CurrentCustomer Row
Options Map
```

8.3 Formatting and masks

Business applications need stable display formatting. Safire should support format masks for dates, times, currency, decimals, percentages, codes, and report fields. Exact mask names may be finalized later, but the source should make formatting visible and owned by the developer.

```
Edit EDT_CreditLimit
    Caption = "Credit Limit"
    Binding = Customer.CreditLimit
    Type = Currency
    Format = "currency"
End

ReportField FLD_InvoiceDate
    Source = Invoice.InvoiceDate
    Format = "yyyy/MM/dd"
End
```

9. Expressions and operators

Expressions combine values, procedure calls, properties, constants, and operators. Safire expressions should remain readable and safe for compiler checking.

9.1 Common operators

Category	Operators / examples
Assignment	=, +=, -=, *=, /=
Arithmetic	+, -, *, /, %, ^
Comparison	=, <>, <, <=, >, >=
Boolean	And, Or, Not
String	+ for concatenation where supported, Format(), Trim(), Left(), Right()
Null checks	IsNull(value), Coalesce(value, fallback)
Membership	In, Contains() where supported

9.2 Expression examples

```
FullName = Trim(Customer.Name)
InvoiceTotal = Subtotal + TaxAmount
CanPost = Active And InvoiceTotal > 0
DisplayName = CustomerCode + " - " + CustomerName
```

```
Allowed = User.Role In ["Admin", "Supervisor"]
DueDate = AddDays(InvoiceDate, PaymentTerms.Days)
```

10. Execution control

Safire control structures should support business logic without unnecessary punctuation. Blocks are explicit and closed with End.

10.1 If / Else

```
If Customer.Active = False Then
    ShowWarning("The customer is inactive.")
Else
    OpenWindow(WIN_CustomerEdit)
End
```

10.2 Case

```
Case Order.Status
    When "New"
        Enable(BTN_Post, True)
    When "Posted"
        Enable(BTN_Post, False)
    Else
        ShowWarning("Unknown order status.")
End
```

10.3 Loops

```
For Each Line In InvoiceLines
    InvoiceTotal += Line.LineTotal
End

While ImportReader.HasMoreRows()
    ImportCustomer(ImportReader.NextRow())
End
```

10.4 Transactions

```
Transaction DB_Main
    SaveRecord(Customer)
    SaveRecord(CustomerContact)
Commit

Try
    PostInvoice(InvoiceID)
Catch ex
    ShowError(ex.Message)
End
```

11. Procedures and functions

Procedures perform actions. Functions return values. Event handlers should usually call procedures and functions rather than containing all business logic inline.

11.1 Procedure syntax

```
Procedure SaveCustomer()
  If ValidateRecord(Customer) = False Then
    ShowValidationSummary(Customer)
    Return
  End

  SaveRecord(Customer)
  Refresh(TBL_Customers)
End
```

11.2 Function syntax

```
Function CustomerDisplayName(CustomerCode String, CustomerName String) Returns String
  Return CustomerCode + " - " + CustomerName
End
```

11.3 Parameter rules

Rule	Recommended behavior
Explicit types	Use explicit parameter and return types for public or shared functions.
Short routines	Keep small private helpers close to the workflow that uses them.
Business names	Use names such as ValidateCustomer, SaveInvoice, OpenCustomerCard.
Avoid hidden globals	Pass context explicitly where it improves testability and AI analysis.
Return errors clearly	Use Result, Error, or exception patterns consistently.

12. Classes and structured types

Classes and structured types should hold reusable business logic, service code, rule sets, adapters, and runtime helpers. Safire classes should support a practical business application model without making simple form logic unnecessarily complex.

12.1 Class example

```
Class CustomerService
  Function IsOverCreditLimit(Customer Row) Returns Boolean
    Return Customer.Balance > Customer.CreditLimit
  End

  Procedure Save(Customer Row)
    ValidateCustomer(Customer)
```

```
    SaveRecord(Customer)
  End
End
```

12.2 Structured type example

```
Type CustomerSummary
  CustomerID Integer
  CustomerCode String
  Name String
  Balance Currency
  Active Boolean
End
```

12.3 Where to use classes

- Business services used by multiple windows.
- Posting and transaction logic.
- Import/export services.
- Validation rule groups.
- Report data preparation.
- Integration adapters.

13. Error handling and validation

Business applications need predictable failure behavior. Safire should make validation, runtime errors, diagnostics, and support evidence understandable to the developer and support team.

13.1 Validation declaration

```
Validation CustomerRules For Table Customer
  Required Customer.CustomerCode Message "Customer code is required."
  Required Customer.Name Message "Customer name is required."
  MinValue Customer.CreditLimit 0 Message "Credit limit may not be negative."
  MaxLength Customer.Email 120 Message "Email address is too long."
End
```

13.2 Runtime validation use

```
On BTN_Save.Click
  If ValidateRecord(Customer) Then
    SaveCustomer()
  Else
    ShowValidationSummary(Customer)
    FocusFirstInvalidField(Customer)
  End
End
```

13.3 Error handling

```
Try
  Transaction DB_Main
    SaveRecord(Customer)
```

```
        WriteAudit("Customer saved", Customer.CustomerID)
    Commit
Catch ex
    RollbackIfActive(DB_Main)
    ShowError(ex.Message)
    LogError(ex)
End
```

13.4 Support diagnostics

A Safire business application should be supportable. Compiler errors, runtime errors, validation summaries, build logs, package manifests, license diagnostics, and support bundles should be clear enough for a developer to diagnose without guessing.

14. Application declaration and startup

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

14.1 Syntax

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

14.2 Example

```
Application ServiceDesk
    Title = "Service Desk"
    Version = "1.0.0"
    Company = "Safire Systems"
    MainWindow = WIN_Main
    DefaultDatabase = DB_Main
    Icon = Resource("icons/service-desk.ico")
    Culture = "en-ZA"
End

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

14.3 Common application properties

Property	Description
Title	Display title of the application.
Version	Application version used for support and packaging.
Company	Publisher or owning company.
MainWindow	First window opened by an interactive desktop

	application.
DefaultDatabase	Default database connection or dictionary root.
Icon	Application icon resource.
Culture	Default locale/culture setting.
LicenseMode	Optional application licensing/runtime mode where applicable.

15. Dictionary, tables, and database source

Safire is intended to be strong in database-backed business applications. Dictionary and table definitions should be visible, source-owned, and useful to designers, browse/update generators, validation, reports, AI analysis, and deployment.

15.1 Table declaration

```
Table Customer
  Field CustomerID Integer Key Auto
  Field CustomerCode String(20) Required Unique Caption "Code"
  Field Name String(120) Required Caption "Customer Name"
  Field Email String(120) Caption "Email"
  Field Phone String(40) Caption "Phone"
  Field CreditLimit Decimal(18,2) Required Default 0
  Field Active Boolean Required Default True
  Field CreatedAt DateTime Required
End
```

15.2 Relationships and lookups

```
Table CustomerContact
  Field ContactID Integer Key Auto
  Field CustomerID Integer Required
  Field ContactName String(120) Required
  Field Email String(120)
  Field Phone String(40)
  Field IsPrimary Boolean Default False

  Relationship CustomerContact.CustomerID References Customer.CustomerID
End

Lookup CustomerLookup
  Table = Customer
  Display = Customer.Name
  Value = Customer.CustomerID
  Sort = Customer.Name
End
```

15.3 Data provider direction

Version 1 developer-preview work includes local data and SQLite/ODBC direction in later runtime planning, but after M651 the important end-user rule is already clear: dictionary and table source should drive data-aware forms, browse lists, validation, queries, reports, and generated update windows.

15.4 Record buffer and CRUD vocabulary

Command / concept	Purpose
NewRecord(Table)	Clear and prepare a new record buffer.
SaveRecord(Table)	Validate and persist the current record.
LoadFirstRecord(Table)	Load the first available record for simple forms/proofs.
DeleteRecord(Table)	Delete the current record where supported.
DeleteAllRecords(Table)	Clear data in controlled test/proof scenarios.
AppendRecord(Table)	Append current buffer to storage where supported.
RecordCount(Table)	Return row count for diagnostics or simple checks.

16. Queries, views, queues, and browse data

Queries and views describe reusable data selections. They are used by browses, reports, lookups, exports, and business services. Queues hold in-memory rows where runtime manipulation, sorting, filtering, or temporary data is useful.

16.1 Query example

```
Query CustomerBrowseView
  From Customer
  Select Customer.CustomerID
  Select Customer.CustomerCode
  Select Customer.Name
  Select Customer.Email
  Select Customer.Phone
  Where Customer.Active = True
  Order By Customer.Name
End
```

16.2 Join query example

```
Query CustomerContactsView
  From Customer
  Join CustomerContact On Customer.CustomerID = CustomerContact.CustomerID
  Select Customer.CustomerCode
  Select Customer.Name
  Select CustomerContact.ContactName
  Select CustomerContact.Email
  Order By Customer.Name, CustomerContact.ContactName
End
```

16.3 Browse data source rules

- A browse should declare its row source explicitly.
- Display columns should map to fields or calculated expressions.
- The selected row should expose its key value for edit, delete, and drill-down actions.
- Sort and search behavior should be visible in source when it affects user behavior.
- Browse refresh should preserve selection where possible after updates.

17. Windows, forms, and source-backed UI

Windows and forms are first-class Safire source artifacts. The source file should describe title, size, window frame behavior, controls, layout, binding, events, and any business-specific metadata needed by the IDE and runtime.

17.1 Window declaration

```
Window CustomerCards
  Title = "Customer Cards"
  Width = 800
  Height = 520
  ShowTitleBar = True
  ShowSystemMenu = True
  ShowCloseButton = True

  Label LBL_Title
    Text = "Customers Maintenance"
    X = 24
    Y = 22
    FontWeight = Bold
  End

  Edit edtCustomerSearch
    X = 160
    Y = 52
    Width = 260
    Placeholder = "Search customers"
  End

  Button BTN_Search
    Text = "Search"
    X = 435
    Y = 52
    OnClick = SearchCustomers
  End
End
```

17.2 M651 designer rule

After M651, the active form designer path should open the actual source surface. If a source form is empty, it should remain blank. If a source form contains controls, the designer should show those controls rather than a hardcoded starter canvas. This is a core source-first behavior.

17.3 Form lifecycle events

Event	Use
Initialization	Prepare runtime state before first display.
Created	Run after controls are created.
Shown	Load data and focus the first useful control.

Hidden	Pause or cleanup visible state.
Destroy	Release resources.
Validate	Validate form data before save or navigation.
Accepted	Handle default accept behavior.

18. Control reference

Safire controls should be business-friendly source declarations mapped to native runtime behavior. The exact catalog will continue to mature, but the following controls represent the important Version 1 business surface.

Control	Business use
Label / StaticText	Captions, headings, explanatory text.
Edit / TextBox	Text, numeric, date, and formatted entry.
Button	Commands such as Save, Search, Print, Post, Cancel.
CheckBox	Boolean choices.
Choice / ComboBox	Lookup selection and constrained values.
ListBox	Simple lists and selections.
Browse / Grid / Table	Business rows with columns, selection, sorting, searching, and update actions.
DatePicker / Calendar	Date entry and date-driven workflows.
Slider / Gauge	Progress or bounded numeric entry where appropriate.
Image	Logos, product images, document previews, and icons.
Panel / GroupBox	Layout grouping and visual organization.
Tabs / Notebook	Multiple pages of related fields.
Menu / Toolbar / StatusBar	Application command surface and status feedback.

18.1 Control example

```
Edit txtCustomerEmail
    Caption = "Email"
    Binding = Customer.Email
    X = 120
    Y = 160
    Width = 240
    Required = False
    Validation = EmailAddress
End

CheckBox chkCustomerActive
    Text = "Active"
    Binding = Customer.Active
    X = 430
    Y = 185
End
```

18.2 Common control properties

Property	Meaning
Name	Stable control identifier used by source, designer, events, and AI.
Text / Caption / Title	User-visible text.
X, Y, Width, Height	Absolute layout values where used.
Dock / Anchor / Sizer	Layout behavior for resizing and containers.
Binding	Data source field, variable, or expression.
Enabled	Whether user can interact with the control.
Visible	Whether the control is shown.
Required	Validation hint for data entry.
Format	Display or entry format.
Tooltip / HelpText	User guidance and help integration.

19. Properties, layout, and visual round-trip

The property grid is a visual editor over source-backed properties. A property change should result in a source change that can be reviewed, written back, reparsed, and shown again in the designer.

19.1 Property write-back principles

- The source file remains the durable truth.
- The designer shows a runtime/design plan derived from source.
- Property edits are converted into a source patch.
- The developer can review the change where appropriate.
- The source is backed up before applying changes.
- The updated source is reparsed to confirm it still describes the design surface.

19.2 Layout choices

Layout style	When to use it
Absolute coordinates	Legacy-style business forms, fixed screens, direct migration, and early proofs.
Anchoring	Forms that resize but retain simple field positions.
Docking	Main windows, panels, output panes, and tool windows.
Sizers / layout containers	Modern resilient layouts and pages that need stable resizing.
Tabs / notebooks	Business forms with grouped pages such as General, Contacts, History, Notes.

19.3 Example container layout

```
Panel PNL_Details
   Layout = Grid
   Columns = 2

   Label LBL_Code
      Text = "Code"
   End
```

```
    Edit EDT_Code
      Binding = Customer.CustomerCode
    End

    Label LBL_Name
      Text = "Customer Name"
    End
    Edit EDT_Name
      Binding = Customer.Name
    End
  End
```

20. Events and event handlers

Events connect user actions and runtime lifecycle points to Safire procedures. Safire event code should be short, clear, and source-backed.

20.1 Event declaration styles

```
On BTN_Save.Click
  SaveCustomer()
End

On txtCustomerEmail.Changed
  MarkDirty(Customer)
End

On CustomerCards.Shown
  LoadCustomers()
  Focus(edtCustomerSearch)
End
```

20.2 Common events

Event group	Examples
Form lifecycle	Initialization, Created, Shown, Hidden, Destroy
Action / validation	Validate, Accepted, Click, DoubleClick, RightClick, ContextMenu
Focus / keyboard	GotFocus, LostFocus, KeyDown, KeyUp, KeyPressed, SystemKeyDown
Text/value	Changed, ValueChanged, CheckedChanged, DateChanged
Selection	SelectionChanged, RowChanged, RowDoubleClick
Menu/toolbar	CommandClick, UpdateUI

20.3 Event handler best practice

- Do not hide business rules in anonymous designer-only code.
- Prefer procedure names that describe intent: SaveCustomer, SearchCustomers, PrintInvoice.
- Keep validation close to data rules, not scattered across many button clicks.
- Log important business actions where auditability matters.

- Use AI assistance to explain or propose changes, not to silently apply event logic.

21. Data binding and browse/update patterns

Safire should make common business UI patterns explicit: browse a set of records, select a row, open an update form, validate, save, refresh the browse, and preserve state.

21.1 Bound edit form example

```
Window WIN_CustomerEdit
  Title = "Customer"
  SourceTable = Customer

  Edit EDT_Code
    Caption = "Code"
    Binding = Customer.CustomerCode
    Required = True
  End

  Edit EDT_Name
    Caption = "Customer Name"
    Binding = Customer.Name
    Required = True
  End

  Button BTN_Save
    Text = "Save"
  End

  On BTN_Save.Click
    If ValidateRecord(Customer) Then
      SaveRecord(Customer)
      CloseWindow()
    Else
      ShowValidationSummary(Customer)
    End
  End
End
```

21.2 Browse form example

```
Browse TBL_Customers
  RowSource = CustomerBrowseView
  KeyField = Customer.CustomerID
  SearchFields = Customer.CustomerCode, Customer.Name, Customer.Email

  Column Customer.CustomerCode Title "Code" Width 90
  Column Customer.Name Title "Customer Name" Width 220
  Column Customer.Email Title "Email" Width 180
  Column Customer.Phone Title "Phone" Width 120
```

```
End

On TBL_Customers.DoubleClick
    OpenWindow(WIN_CustomerEdit, SelectedKey(TBL_Customers))
End
```

21.3 Update pattern flow

1. Browse query selects rows.
2. User searches or filters.
3. User selects a row.
4. Edit window loads the selected record.
5. Controls bind to table fields.
6. Validation runs before save.
7. Save runs inside the appropriate data provider path.
8. Browse refreshes and attempts to retain selection.
9. Dirty-state guards prevent accidental data loss.

22. Reports and printable output

Reports are first-class Safire application artifacts. A report should be source-backed, visually editable, previewable, printable, exportable, and callable from application code.

22.1 Report declaration

```
Report RPT_CustomerList
    Title = "Customer List"
    Source = CustomerBrowseView
    PageSize = A4
    Orientation = Portrait
    Margins = 12mm, 12mm, 12mm, 12mm

    Header
        Label "Customer List" X=0 Y=0 Width=180 Height=8 FontSize=14 Bold=True
    End

    Detail
        Field Customer.CustomerCode X=0 Y=0 Width=25 Height=5
        Field Customer.Name X=30 Y=0 Width=70 Height=5
        Field Customer.Email X=105 Y=0 Width=60 Height=5
    End

    Footer
        PageNumber Format "Page {Page} of {Pages}"
    End
End
```

22.2 Report designer behavior

- Report bands, fields, labels, expressions, totals, grouping, and page setup belong in source-backed report definitions.
- Visual editing should write back source rather than hiding report truth in designer state.
- Preview, print, PDF, and export paths should be reproducible from report source.
- Validation should catch missing data sources, invalid expressions, overlapping controls, and output problems where possible.

22.3 Calling a report

```
Procedure PrintCustomerList()
  ReportOptions Options
  Options.Output = Preview
  Options.Parameters["ActiveOnly"] = True
  RunReport(RPT_CustomerList, Options)
End
```

23. AI-aware source and the SafireIDE AI Assistant Pad

Safire AI is intended to be a RAD-aware project assistant, not an uncontrolled coding chatbot. It should understand project structure, source files, UI, dictionaries, reports, queries, diagnostics, and build output.

23.1 Safe AI workflow

1. The developer asks a question or requests a change.
2. Safire builds project context from source and diagnostics.
3. The AI proposes an explanation or change package.
4. The developer reviews the proposal.
5. Safire shows a diff for source changes.
6. The developer approves apply.
7. Safire creates a backup and audit entry.
8. Safire runs build/test/proof steps where appropriate.
9. The developer can restore or roll back if necessary.

23.2 Source style that helps AI

- Use stable names for windows, controls, tables, fields, reports, and procedures.
- Keep event handlers short and call named procedures.
- Write comments that explain business rules and unusual customer requirements.
- Separate generated output from source.
- Keep validation declarations close to table or form rules.
- Avoid clever one-line logic where clarity is more important.

23.3 Useful AI prompts inside SafireIDE

Prompt	Expected result
Explain this form.	AI summarizes controls, bindings, events, validation, and related source files.

Find where Customer.CreditLimit is used.	AI returns relevant forms, reports, procedures, queries, and validation rules.
Propose a required-field validation for Customer.Email.	AI prepares a source patch proposal and explains the impact.
Why did the build fail?	AI reads diagnostics and points to likely source causes.
Create a customer browse and update form from this dictionary.	AI proposes generated source and wizard steps for developer approval.

24. Build, run, package, and deployment

The compiler and runtime path is central to Safire. The IDE is a productivity layer over a real language, source model, and build pipeline. A deployed end-user application should not require the SafireIDE.

24.1 Build lifecycle

1. Load project manifest.
2. Read source folders and owned artifacts.
3. Parse applications, windows, reports, tables, dictionaries, queries, procedures, and classes.
4. Run semantic checks and validation.
5. Generate runtime plans and required intermediate artifacts.
6. Compile or package executable targets.
7. Stage runtime dependencies and resources.
8. Run smoke or regression checks where configured.
9. Create release or deployment package.

24.2 Build/run commands

Command area	Purpose
Build Project	Build the active project executable or package target.
Run Active Project EXE / F5	Launch the generated application executable.
Validate	Run source, form, report, dictionary, and project checks.
Preview	Preview form/report/runtime output without full deployment.
Package	Create deployable application output for customers or testers.
Support bundle	Collect diagnostics, logs, manifests, and environment details for support.

24.3 Deployment rule

End users receive the compiled application, runtime components, configuration, database drivers, report/export runtime where needed, resources, and support tools. They do not receive or need the licensed SafireIDE for ordinary application use.

25. Licensing and product boundary

Licensing is part of the Safire trust model. The professional IDE is a licensed productivity product. The source model and build path should still give developers confidence that they remain in control of their applications.

25.1 Recommended product boundary

Capability	Recommended position
Safire source language	Developer-owned project source; readable and maintainable.
Compiler / CLI path	Available enough for checking and building without forcing every maintenance step through the licensed IDE.
Runtime and deployed apps	Independent from the licensed IDE; customers run deployed applications without SafireIDE.
SafireIDE	Licensed professional productivity product.
Visual designers	Licensed productivity features.
AI Assistant Pad	Licensed IDE feature because it provides project-aware productivity, patch review, audit, diagnostics, and integrated workflow.
Support tools and diagnostics	Part of commercial support and licensing workflow where appropriate.

25.2 Developer trust statement

Safire should be commercial without trapping the developer. The licensed IDE should be worth paying for because it saves time, improves visibility, controls AI changes, provides visual designers, and integrates build/run/support workflows. It should not be valuable only because the developer is locked out without it.

26. Complete example: Customer manager

This example shows a compact business application with an application declaration, table, query, browse, edit form, validation, and save logic.

```
Project CustomerManager
    Name = "Customer Manager"
    Version = "1.0.0"
    Application = app/Application.sf
    SourceFolders = app, windows, reports, dictionaries, queries, procedures
    OutputFolder = build/output
End

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

Table Customer
```

```
  Field CustomerID Integer Key Auto
  Field CustomerCode String(20) Required Unique Caption "Code"
  Field Name String(120) Required Caption "Customer Name"
  Field Email String(120)
  Field Phone String(40)
  Field CreditLimit Decimal(18,2) Default 0
  Field Active Boolean Default True
End

Query CustomerBrowseView
  From Customer
  Select Customer.CustomerID
  Select Customer.CustomerCode
  Select Customer.Name
  Select Customer.Email
  Select Customer.Phone
  Where Customer.Active = True
  Order By Customer.Name
End

Window WIN_CustomerBrowse
  Title = "Customers"
  Width = 900
  Height = 600

  Browse TBL_Customers
    RowSource = CustomerBrowseView
    KeyField = Customer.CustomerID
    Column Customer.CustomerCode Title "Code" Width 90
    Column Customer.Name Title "Customer Name" Width 220
    Column Customer.Email Title "Email" Width 180
    Column Customer.Phone Title "Phone" Width 120
  End

  Button BTN_New Text "New"
  Button BTN_Edit Text "Edit"
  Button BTN_Close Text "Close"

  On BTN_New.Click
    OpenWindow(WIN_CustomerEdit, NewRecord(Customer))
  End

  On BTN_Edit.Click
    OpenWindow(WIN_CustomerEdit, SelectedKey(TBL_Customers))
  End
End

Window WIN_CustomerEdit
```

```
    Title = "Customer"
    SourceTable = Customer

    Edit EDT_Code Binding Customer.CustomerCode Required True
    Edit EDT_Name Binding Customer.Name Required True
    Edit EDT_Email Binding Customer.Email
    Edit EDT_Phone Binding Customer.Phone
    Edit EDT_CreditLimit Binding Customer.CreditLimit Format "currency"
    CheckBox CHK_Active Binding Customer.Active Text "Active"

    Button BTN_Save Text "Save"
    Button BTN_Cancel Text "Cancel"

    On BTN_Save.Click
        If ValidateRecord(Customer) Then
            SaveRecord(Customer)
            CloseWindow()
        Else
            ShowValidationSummary(Customer)
        End
    End

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

27. Complete example: report and query

This example adds a report to the customer manager application. The report uses a query as its row source and can be called from a button or menu command.

```
Query CustomerListReportData
    From Customer
    Select Customer.CustomerCode
    Select Customer.Name
    Select Customer.Email
    Select Customer.Phone
    Select Customer.Active
    Order By Customer.Name
End

Report RPT_CustomerList
    Title = "Customer List"
    Source = CustomerListReportData
    PageSize = A4
    Orientation = Portrait

    Header
```

```
    Label "Customer List" X=0 Y=0 Width=180 Height=8 FontSize=14 Bold=True
    Line X=0 Y=10 Width=180
  End

  Detail
    Field Customer.CustomerCode X=0 Y=0 Width=25 Height=5
    Field Customer.Name X=28 Y=0 Width=60 Height=5
    Field Customer.Email X=90 Y=0 Width=55 Height=5
    Field Customer.Phone X=148 Y=0 Width=32 Height=5
  End

  Footer
    PageNumber Format "Page {Page}"
  End
End

Procedure PreviewCustomerList()
  RunReport(RPT_CustomerList, Preview)
End
```

28. Troubleshooting and developer practices

28.1 Designer shows wrong surface

- Confirm the active tab is the intended .sfform file.
- Save the form and reopen it to force source reload.
- Check whether the source file is empty or contains controls.
- After M651, an empty form should remain blank. It should not display a starter customer canvas unless that content exists in the source.
- Check the output panel for parser or runtime-plan messages.

28.2 Build fails

- Read the first compiler diagnostic, not only the last line.
- Check for source write-back that left a partial block or missing End.
- Run validation before build if available.
- Use the AI Assistant Pad to summarize the error and identify related source files.
- Restore the previous backup if the last approved change caused the failure.

28.3 AI proposal looks too broad

- Ask the AI Pad to reduce the proposal to the smallest safe patch.
- Review the diff before apply.
- Require backup before apply.
- Build after apply.
- Record the reason in audit notes where appropriate.

28.4 Source discipline

- Keep source under version control.
- Avoid editing generated files unless intentionally promoting them to source.

- Use one release folder policy when staging release output if that is the project standard.
- Keep project folders portable.
- Do not depend on absolute local paths for runtime behavior.
- Document customer-specific business rules in source comments or docs.

29. Appendices

Appendix A: quick reference - common procedures

Procedure	Purpose
OpenWindow(WindowName)	Open a source-defined window.
CloseWindow()	Close the active window.
SetText(Control, Text)	Set user-visible text on a control.
GetText(Control)	Read text from a control.
SetValue(Control, Value)	Set a control value.
Enable(Control, Boolean)	Enable or disable a control.
Refresh(Control)	Refresh a browse, list, or display control.
Message(Text)	Show an informational message.
ShowWarning(Text)	Show a warning.
ShowError(Text)	Show an error.
ValidateRecord(Table)	Run validation rules for a record.
SaveRecord(Table)	Persist the current record.
RunReport(Report, Options)	Run a source-defined report.
Log(Text)	Write diagnostic or audit output.

Appendix B: quick reference - source artifacts

Artifact	Owned by developer?	Notes
Project manifest	Yes	Defines project structure and build metadata.
Application source	Yes	Defines startup and application-level metadata.
Window/form source	Yes	Source-backed design surface.
Report source	Yes	Source-backed printable output.
Dictionary/table source	Yes	Defines data model and validation metadata.
Query source	Yes	Defines reusable selections.
Procedure/class source	Yes	Defines business logic.
Generated runtime plan	No	Derived from source.
Build logs	No	Diagnostic evidence.
Preview files	No	Derived output.

Appendix C: M651 checklist for source-backed forms

1. Open a real .sfform from the project tree.
2. Confirm the designer displays the actual source controls.
3. Open a blank .sfform and confirm it remains blank.
4. Change a property and save/write back.

5. Reopen the form and confirm the property persists.
6. Add or select a control and confirm the property grid reflects the selected source control.
7. Confirm events are displayed from the event catalog.
8. Build and run the active project when the form is part of the application path.

Appendix D: glossary

Term	Meaning
Source-first	The durable application truth is in developer-owned source files.
Runtime plan	Generated structure used by the IDE/runtime to preview or build from source.
Designer round-trip	The cycle of reading source, showing the design surface, editing visually, and writing back source.
AI proposal	A suggested change prepared for review, not directly applied without approval.
Browse/update	Standard business pattern for listing records and editing one selected record.
Dictionary	Source-owned data model metadata: tables, fields, keys, relationships, validation, and lookups.
Licensed IDE	The professional productivity product including visual designers, AI pad, integrated build/run, and support tooling.
Non-IDE path	Text editor plus command-line tools sufficient for trust, build, and maintenance without full IDE productivity.

Closing note

Safire after M651 should be understood as a serious source-first business application development system in developer preview. Its strongest promise is not only visual RAD productivity, but developer control: readable source, visible design state, buildable projects, deployable applications, licensed productivity features, and AI assistance that stays inside review, backup, and audit boundaries.