Safire1 Developer Documentation Set

Version 1 Alpha Reference Manual

Document set	Language, project, runtime, database, UI, report, and debugging reference
Audience	Developers building Safire business applications
Source model	Source-first, readable, buildable, inspectable, and deployable
Delivery format	PDF reference manual

This manual is written as a developer reference for the Safire1 Version 1 Alpha line. It describes the developer-facing language and tooling contract in the same direct reference style used by technical language manuals: syntax, parameters, behavior, examples, related topics, and diagnostic guidance.

> The manual is self-contained. It documents Safire concepts directly and does not require the reader to know another development product.

Table of Contents

Chapter 1 - Introduction and Documentation Conventions

Purpose of this manual

This manual documents the Safire1 developer surface as a practical reference. It is not a beginner programming textbook. It is intended for developers who need precise answers about source files, declarations, project files, build behavior, runtime services, windows, reports, database drivers, thread behavior, internationalisation, and diagnostics.

Safire is a source-first business application system. The application design, project structure, windows, reports, data dictionaries, procedures, classes, queries, validation rules, and build settings are represented as developer-owned source artifacts. Designers and tooling are productivity layers over that source model.

Reference item format

Language and runtime items are documented with a repeated pattern so that each topic can be read independently. The common form is shown below.

```
ITEM-NAME (short description)

Syntax
    ITEM-NAME parameter [optional-parameter]

Parameters
    parameter            Required value or identifier.
    optional-parameter   Optional value.

Description
    Concise behavior description.

Example
    ITEM-NAME ExampleValue

See also
    Related Safire items.
```

Syntax notation

Symbol	Meaning
[item]	The item is optional.
item | item	One of the alternatives is selected.
...	The previous item may repeat.
identifier	A developer-supplied name.
literal	A fixed value written directly in source.
END	Terminates a structured declaration or executable block.

Status labels

Label	Meaning
Implemented	The item is expected to be present in the current Safire1 toolchain.
Alpha contract	The item is the current developer-facing contract for Version 1 Alpha and may still receive tightening.
Reserved	The name or structure is reserved for the Version 1 family but may not yet be fully operational.
Design rule	A product rule that tooling and source should preserve.

Common source artifact names

Artifact	Purpose
.sf	Safire source module.
.sfproj	Canonical project manifest in JSON format.
.sfsol	Canonical solution/workspace file that can reference one or more projects.
.sfdict	Dictionary and database metadata source file.
.sfwin	Window/form source artifact when separated from general source.
.sfreport	Report source artifact when separated from general source.
build/	Disposable build output.
metadata/	Generated indexes, build handoff files, and tool-readable summaries.
release/	Deployable runtime output when a project is packaged.

Safire tools may support consolidated source files or split source files. Split files are preferred for larger applications because they make designer write-back, review, and version-control diffs easier.

Chapter 2 - Safire Language Reference

Language overview

Safire source is organized around business application artifacts: applications, procedures, functions, classes, records, tables, queries, windows, controls, events, reports, validation rules, and runtime services. Source is readable text and is intended to remain useful in a standard editor as well as inside the Safire IDE.

Safire is statement-oriented at the outer declaration level. A declaration normally begins with a keyword, has named parameters or attributes, and ends with END when it owns nested content. Executable code uses structured blocks for IF, CASE, LOOP, FOR EACH, TRY, and ON event handlers.

Program structure

```
APPLICATION CustomerManager
  TITLE "Customer Manager"
  START MainWindow
  USE DICTIONARY "data/CustomerDictionary.sfdict"
END

PROCEDURE Main
CODE
  OpenWindow(MainWindow)
END
```

An APPLICATION declaration names the application, defines the startup window or procedure, attaches dictionaries and resources, and supplies deployment metadata. One project may contain multiple modules, but exactly one startup application is selected by the active build target.

Identifiers

Identifiers name applications, procedures, classes, fields, controls, reports, events, and resources. Identifiers should begin with a letter or underscore, followed by letters, digits, or underscores. Dots are used for qualified access, not as ordinary identifier characters.

Rule	Description
Case preservation	The compiler preserves written casing for display and diagnostics.
Case lookup	Project policy may choose strict or relaxed lookup. Strict lookup is recommended for shared teams.
Reserved words	Reserved keywords cannot be used as identifiers unless a future escape mechanism explicitly permits it.
Qualified names	Use Owner.Member to reference a member inside a structured artifact.
Designer names	Designer-created controls should use stable names, not temporary captions.

Comments

A line comment begins with an exclamation mark and continues to the end of the line. Documentation comments begin with three exclamation marks and are intended for help generation, AI context, and hover help in the IDE.

```
! This is a normal comment.
!!! Opens the customer edit window and returns Accepted when saved.
PROCEDURE EditCustomer(CustomerId : Int) RETURNS DialogResult
```

Core declaration keywords

APPLICATION (declare a runnable application)

Item	Description
Status	Alpha contract
Purpose	declare a runnable application

Syntax

```
APPLICATION identifier
  TITLE string
  START procedure-or-window
  [USE DICTIONARY path]
  [ICON path]
  [VERSION string]
END
```

Parameter	Meaning
identifier	Application name.
TITLE	User-facing title.
START	Startup procedure or window.

Notes: APPLICATION owns application-level metadata and startup behavior.

Example

```
APPLICATION Invoicing
  TITLE "Invoicing"
  START InvoiceMain
  VERSION "1.0.0"
END
```

See also: PROJECT, WINDOW, PROCEDURE

MODULE (declare a source module)

Item	Description
Status	Alpha contract
Purpose	declare a source module

Syntax

```
MODULE identifier
  [USES module-list]
  declarations
END
```

Parameter	Meaning
identifier	Module name.
USES	Optional list of imported modules.

Notes: MODULE groups declarations and allows larger projects to be organized into maintainable units.

Example

```
MODULE CustomerProcedures
  USES CustomerTypes, CustomerWindows
END
```

See also: INCLUDE, PROCEDURE, CLASS

INCLUDE (include another source file)

Item	Description
Status	Alpha contract
Purpose	include another source file

Syntax

```
INCLUDE "relative/path/to/file.sf" [ONCE]
```

Parameter	Meaning
path	Project-relative source path.
ONCE	Prevents repeated inclusion.

Notes: INCLUDE is intended for shared declarations and generated source that is deliberately part of the source model.

Example

```
INCLUDE "shared/Validation.sf" ONCE
```

See also: MODULE, PROJECT

DEFINE (declare a compile-time value)

Item	Description
Status	Alpha contract
Purpose	declare a compile-time value

Syntax

```
DEFINE identifier = literal
```

Parameter	Meaning
identifier	Name of the compile-time value.
literal	String, number, true, false, or project-defined literal.

Notes: DEFINE creates a named value available during compilation and build planning.

Example

```
DEFINE AppName = "Customer Manager"
```

See also: CONST, PROJECT

CONST (declare a constant value)

Item	Description
Status	Alpha contract
Purpose	declare a constant value

Syntax

```
CONST identifier : Type = expression
```

Parameter	Meaning
identifier	Constant name.
Type	Safire data type.
expression	Compile-time expression.

Notes: CONST creates a typed value that cannot be reassigned.

Example

```
CONST MaxRecentFiles : Int = 10
```

See also: VAR, DEFINE

Procedures and functions

Procedures perform actions. Functions return values. Both may declare parameters, local variables, validation rules, error handling, and runtime service calls. A procedure or function may be private to a module or exported as part of a project library.

PROCEDURE (declare executable action)

Item	Description
Status	Alpha contract
Purpose	declare executable action

Syntax

```
PROCEDURE identifier([parameter-list]) [RETURNS Type]
  LOCAL
    declarations
CODE
  statements
END
```

Parameter	Meaning
identifier	Procedure name.
parameter-list	Zero or more typed parameters.
RETURNS	Optional return type.

Notes: A PROCEDURE may return a value when RETURNS is supplied. Use RETURN to exit with a value.

Example

```
PROCEDURE FullName(FirstName : String, LastName : String) RETURNS String
CODE
  RETURN Trim(FirstName) + " " + Trim(LastName)
END
```

See also: FUNCTION, RETURN, LOCAL

FUNCTION (declare value-returning routine)

Item	Description
Status	Alpha contract
Purpose	declare value-returning routine

Syntax

```
FUNCTION identifier([parameter-list]) RETURNS Type
  LOCAL
    declarations
CODE
  statements
END
```

Parameter	Meaning
identifier	Function name.
RETURNS	Required return type.

Notes: FUNCTION is used when the routine is primarily evaluated for its return value.

Example

```
FUNCTION IsCreditHold(Customer : CustomerRecord) RETURNS Bool
CODE
  RETURN Customer.Balance > Customer.CreditLimit
END
```

See also: PROCEDURE, RETURN

LOCAL (begin local declaration section)

Item	Description
Status	Alpha contract
Purpose	begin local declaration section

Syntax

```
LOCAL
  declarations
```

Parameter	Meaning
declarations	Variable, cursor, object, or service references local to the routine.

Notes: LOCAL declarations are created for each routine invocation and are not visible outside the routine.

Example

```
LOCAL
  Total : Decimal(12,2) = 0.00
  Found : Bool = false
```

See also: VAR, PROCEDURE

RETURN (exit a routine)

Item	Description
Status	Alpha contract
Purpose	exit a routine

Syntax

```
RETURN [expression]
```

Parameter	Meaning
expression	Required when the routine returns a value.

Notes: RETURN exits the current procedure or function. In cleanup-sensitive code, FINALLY blocks still run.

Example

```
IF CustomerId = 0
  RETURN ErrorResult("Missing customer")
END
```

See also: TRY, FUNCTION

Classes and records

Structured types group fields and behavior. RECORD is normally used for data rows, messages, and transfer objects. CLASS is used for behavior-rich components with methods and state.

RECORD (declare a structured value)

Item	Description
Status	Alpha contract
Purpose	declare a structured value

Syntax

```
RECORD identifier
  field declarations
END
```

Parameter	Meaning
identifier	Record type name.
field declarations	Typed fields.

Notes: RECORD values are intended for row-shaped or message-shaped data.

Example

```
RECORD CustomerRecord
  Id          : Int
  Name        : String(120)
  Balance     : Decimal(12,2)
  Active      : Bool = true
END
```

See also: TABLE, CLASS, VAR

CLASS (declare a behavior type)

Item	Description
Status	Alpha contract
Purpose	declare a behavior type

Syntax

```
CLASS identifier [EXTENDS base-class]
  FIELD declarations
  METHOD declarations
END
```

Parameter	Meaning
identifier	Class type name.
EXTENDS	Optional base type.

Notes: CLASS groups fields and methods. Public methods form the class surface.

Example

```
CLASS CustomerRules
  METHOD CanInvoice(Customer : CustomerRecord) RETURNS Bool
END
```

See also: METHOD, RECORD

METHOD (declare class behavior)

Item	Description
Status	Alpha contract
Purpose	declare class behavior

Syntax

```
METHOD ClassName.MethodName([parameter-list]) RETURNS Type
CODE
  statements
END
```

Parameter	Meaning
ClassName	Owning class.
MethodName	Method name.

Notes: METHOD is called through an instance or service reference.

Example

```
METHOD CustomerRules.CanInvoice(Customer : CustomerRecord) RETURNS Bool
CODE
  RETURN Customer.Active AND Customer.Balance <= Customer.CreditLimit
END
```

See also: CLASS, PROCEDURE

Execution control

IF (conditional execution)

Item	Description
Status	Alpha contract
Purpose	conditional execution

Syntax

```
IF condition
  statements
[ELSIF condition
  statements]
[ELSE
  statements]
END
```

Parameter	Meaning
condition	Boolean expression.

Notes: IF selects a block based on a Boolean expression.

Example

```
IF Amount > CreditLimit
  ShowMessage("Credit limit exceeded")
ELSE
  SaveInvoice()
END
```

See also: CASE, BOOL

CASE (multi-branch selection)

Item	Description
Status	Alpha contract
Purpose	multi-branch selection

Syntax

```
CASE expression
OF value
  statements
[OF value
  statements]
[ELSE
  statements]
END
```

Parameter	Meaning
expression	Value being tested.
value	Literal or constant branch value.

Notes: CASE chooses the first matching branch. ELSE is optional.

Example

```
CASE Command
OF "New"
  NewCustomer()
OF "Delete"
  DeleteCustomer()
ELSE
  ShowMessage("Unknown command")
END
```

See also: IF

LOOP (repeating block)

Item	Description
Status	Alpha contract
Purpose	repeating block

Syntax

```
LOOP [WHILE condition]
  statements
  [BREAK]
  [CYCLE]
END
```

Parameter	Meaning
condition	Optional loop condition.

Notes: LOOP repeats until BREAK exits or the optional condition becomes false.

Example

```
LOOP WHILE Reader.Next()
  Total += Reader.Amount
END
```

See also: FOR EACH, BREAK

FOR EACH (iterate a collection)

Item	Description
Status	Alpha contract
Purpose	iterate a collection

Syntax

```
FOR EACH item IN collection
  statements
END
```

Parameter	Meaning
item	Loop variable.
collection	Collection, cursor, or enumerable service.

Notes: FOR EACH iterates the current contents or stream supplied by the collection.

Example

```
FOR EACH Line IN Invoice.Lines
  Total += Line.Quantity * Line.Price
END
```

See also: LOOP, CURSOR

TRY (error handling block)

Item	Description
Status	Alpha contract
Purpose	error handling block

Syntax

```
TRY
  statements
CATCH errorName
  statements
[FINALLY
  statements]
END
```

Parameter	Meaning
errorName	The trapped error object.

Notes: TRY traps errors raised inside the block. FINALLY runs whether the block succeeds or fails.

Example

```
TRY
```

```
  Customers.Save(Customer)
CATCH Err
  LogError(Err)
  ShowMessage(Err.Message)
END
```

See also: RAISE, ERROR

Expressions and operators

Group	Operators	Meaning
Arithmetic	+ - * / %	Numeric calculation.
Comparison	= <> < <= > >=	Value comparison.
Logical	AND OR NOT	Boolean combination.
Assignment	= += -= *= /=	Set or update a variable.
Text	+	Concatenate text values.
Qualification	.	Access a member of a structure, class, module, window, or report.
Indexing	[ ]	Access a list, array, queue, or collection element.

Operator precedence is intentionally conventional and should be made explicit with parentheses when the expression is business-critical. Code reviewers should prefer clarity over compactness.

Chapter 3 - Safire Internationalisation

Internationalisation model

Safire source is Unicode text. Internationalisation is based on stable resource keys, translation catalogs, locale-aware formatting, and separation between user-facing text and business logic. A project should not hard-code user-visible text in validation, windows, reports, or error messages unless the text is temporary developer-only output.

The runtime resolves text by key at display time. Designers should store a resource key and a default text value so that screens remain readable before a translation catalog is complete.

Resource file structure

```
RESOURCE CATALOG "resources/text.en.sfres"
  TEXT App.Title = "Customer Manager"
  TEXT Customer.Name.Label = "Name"
  TEXT Customer.Save.Success = "Customer saved"
  TEXT Error.Required = "{FieldName} is required"
END
```

Resource type	Purpose
TEXT	User-facing text and messages.
FORMAT	Number, date, time, currency, and measurement formatting rules.
IMAGE	Locale-specific image or icon references when necessary.
HELP	Context-sensitive help topics.
REPORTTEXT	Report titles, labels, headings, captions, and footers.

Internationalisation keywords

TEXTKEY (refer to translated text)

Item	Description
Status	Alpha contract
Purpose	refer to translated text

Syntax

```
TEXTKEY "resource.key" [DEFAULT "fallback text"]
```

Parameter	Meaning
resource.key	Stable key in a resource catalog.
DEFAULT	Fallback text used when the key is missing.

Notes: TEXTKEY is preferred for captions, labels, messages, menu items, and report text.

Example

```
BUTTON SaveButton
  TEXT TEXTKEY "Customer.Save" DEFAULT "Save"
END
```

See also: RESOURCE, FORMATVALUE

FORMATVALUE (format data for display)

Item	Description
Status	Alpha contract
Purpose	format data for display

Syntax

```
FORMATVALUE(value, format-key [, locale])
```

Parameter	Meaning
value	Value to format.
format-key	Named formatting rule.
locale	Optional locale override.

Notes: FORMATVALUE formats data without changing the stored value.

Example

```
DisplayBalance = FORMATVALUE(Customer.Balance, "Money.Standard")
```

See also: PARSEVALUE, RESOURCE

PARSEVALUE (parse localized input)

Item	Description
Status	Alpha contract
Purpose	parse localized input

Syntax

```
PARSEVALUE(text, type, format-key [, locale])
```

Parameter	Meaning
text	User-entered text.
type	Target Safire type.
format-key	Expected format rule.

Notes: PARSEVALUE converts localized user input into a typed value and raises a validation error if the input is invalid.

Example

```
InvoiceDate = PARSEVALUE(DateEntry.Text, Date, "Date.Short")
```

See also: FORMATVALUE, VALIDATE

LOCALE (declare project locale behavior)

Item	Description
Status	Alpha contract
Purpose	declare project locale behavior

Syntax

```
LOCALE identifier
  LANGUAGE "language-code"
  REGION "region-code"
  DECIMAL_SEPARATOR string
  DATE_ORDER order
END
```

Parameter	Meaning
identifier	Locale name.
LANGUAGE	Language code.
REGION	Region code.

Notes: LOCALE is a project-level declaration for testing, packaging, and resource selection.

Example

```
LOCALE EnglishZA
  LANGUAGE "en"
  REGION "ZA"
```

```
  DECIMAL_SEPARATOR "."
  DATE_ORDER "YMD"
END
```

See also: RESOURCE, PROJECT

Translation workflow

1. Develop with TEXTKEY and DEFAULT values in source.
2. Run the resource extractor to collect keys from windows, reports, validation rules, and procedures.
3. Translate catalog files without changing source logic.
4. Run the missing-key audit before packaging.
5. Test the application with each supported locale and verify screen sizing, report headings, and sorted lists.

Internationalisation diagnostics

Code	Meaning	Recommended action
SFI0001	Resource catalog missing.	Verify project resource paths.
SFI0002	Resource key not found.	Add the key or provide a DEFAULT value.
SFI0003	Text does not fit designed control bounds.	Resize the control or enable auto layout.
SFI0004	Localized value parse failed.	Check the active format rule and input mask.
SFI0005	Catalog contains duplicate key.	Keep one definition per locale catalog.

Chapter 4 - Safire Project and Build System

Project system overview

The Safire project system is source-first and JSON-backed. The canonical project file extension is .sfproj. The canonical solution/workspace extension is .sfsol. Project and solution files are developer-owned source artifacts and should remain readable, reviewable, and stable across machines.

> Build output, indexes, designer caches, runtime plans, and temporary files are derived artifacts. They may be deleted and regenerated from source. They must not become the only location of application truth.

Recommended project folder layout

```
CustomerManager/
  CustomerManager.sfsol
  src/
    App.sf
    windows/
    reports/
    procedures/
    classes/
  dictionary/
    CustomerDictionary.sfdict
  resources/
    text.en.sfres
    images/
  tests/
  config/
  build/
  metadata/
  release/
```

.sfsol solution file

```
{
  "safireSolutionVersion": 1,
  "name": "CustomerManager",
  "projects": [
    { "path": "CustomerManager.sfproj", "role": "application" }
  ],
  "activeProject": "CustomerManager.sfproj"
}
```

.sfproj project file

```
{
  "safireProjectVersion": 1,
  "name": "CustomerManager",
  "kind": "application",
  "main": "src/App.sf",
  "sourceRoots": ["src"],
  "dictionary": ["dictionary/CustomerDictionary.sfdict"],
  "resources": ["resources/text.en.sfres"],
  "build": {
    "configuration": "Debug",
    "outputName": "CustomerManager",
    "runtime": "SafireVM",
    "targets": ["bytecode", "nativeExe"],
    "warningsAsErrors": false
  }
}
```

Project manifest fields

Field	Required	Description
safireProjectVersion	Yes	Project schema version.
name	Yes	Stable project name.
kind	Yes	application, library, service, test, or package.
main	Yes for application	Startup source file.
sourceRoots	Yes	Project-relative folders scanned for source.
dictionary	No	Dictionary source files.
resources	No	Resource catalogs and assets.
build	Yes	Build target and output settings.
dependencies	No	Other project or runtime packages.
tooling	No	IDE and assistant configuration that is safe to store in source.

Build pipeline

1. Read the solution and active project.
2. Normalize project-relative paths.
3. Discover source files from sourceRoots and explicit references.
4. Parse source and resource catalogs.
5. Perform semantic analysis and symbol resolution.
6. Write compiler diagnostics and metadata summaries.
7. Emit bytecode and runtime metadata.
8. Optionally emit native application artifacts for the selected target.
9. Collect runtime dependencies and database/report support files.
10. Write release packaging output if packaging is requested.

Command-line tools

Tool	Purpose
safirec.exe	Compiler and semantic checker.
SafireRun.exe	Runtime launcher for bytecode and runtime metadata.
SafireBuild.exe	Project-aware build, package, clean, and diagnostic tool.
SafireIDE.exe	Developer IDE and design surface launcher.

```
SafireBuild.exe build CustomerManager.sfproj
SafireBuild.exe clean CustomerManager.sfproj
SafireBuild.exe package CustomerManager.sfproj --configuration Release
safirec.exe check src/App.sf --project CustomerManager.sfproj
SafireRun.exe build/CustomerManager.sfbc
```

Build diagnostics and logs

Every build should write a concise console summary and a full log under build/logs. Tooling should avoid silent failure. The summary should state the project path, configuration, target, output path, diagnostics count, and pass/fail result.

Log file	Purpose
build/logs/compiler.check.txt	Semantic check result and symbol summary.

Log file	Purpose
build/logs/build.out.txt	Standard build output.
build/logs/build.err.txt	Build errors and warnings.
build/logs/package.manifest.json	Packaged files and runtime dependencies.
metadata/build/last-build.json	Machine-readable last build handoff.

Build-system reference items

PROJECT (declare project metadata in source)

Item	Description
Status	Alpha contract
Purpose	declare project metadata in source

Syntax

```
PROJECT identifier
  FILE "CustomerManager.sfproj"
  CONFIGURATION "Debug" | "Release"
END
```

Parameter	Meaning
identifier	Project alias in source.
FILE	Project manifest path.

Notes: PROJECT is optional in ordinary modules because the .sfproj file is the authority, but it is useful for generated documentation and diagnostics.

Example

```
PROJECT CustomerManager
  FILE "CustomerManager.sfproj"
END
```

See also: .sfproj, SafireBuild

TARGET (select build output target)

Item	Description
Status	Alpha contract
Purpose	select build output target

Syntax

```
TARGET bytecode | nativeExe | library | package
```

Parameter	Meaning
bytecode	Safire VM runtime output.
nativeExe	Native application output.
library	Reusable project output.
package	Deployable package.

Notes: TARGET may be specified in the project manifest or supplied on the command line.

Example

```
SafireBuild.exe build CustomerManager.sfproj --target bytecode
```

See also: BUILD, PACKAGE

RESOURCE (attach resources to a project)

Item	Description
Status	Alpha contract
Purpose	attach resources to a project

Syntax

```
RESOURCE path [TYPE type-name]
```

Parameter	Meaning
path	Project-relative resource path.
TYPE	Optional resource type.

Notes: RESOURCE links catalogs, images, help, report assets, and other project-owned files.

Example

```
RESOURCE "resources/text.en.sfres" TYPE TextCatalog
```

See also: TEXTKEY, PROJECT

Chapter 5 - Safire Thread Model

Threading overview

The Safire thread model separates the UI thread, background worker threads, runtime service threads, and database connection contexts. UI objects are owned by the UI thread. Background work communicates through posted messages, task results, cancellation tokens, and synchronized data structures.

A Safire application should treat window and control objects as UI-thread owned. Background threads may compute, read files, query data services, or prepare report data, but they must not directly mutate visible controls.

Thread categories

Thread category	Purpose	UI access
UI thread	Owns windows, controls, menus, timers, and visual events.	Direct access allowed.
Worker thread	Runs background jobs, imports, exports, long calculations, or server calls.	Use POST TO UI.
Runtime service thread	Owned by Safire runtime for internal dispatch or scheduling.	No direct application access unless documented.
Database task context	Runs database operations with a connection or transaction context.	Return results through task completion.

Thread reference items

THREAD (declare a thread procedure)

Item	Description
Status	Alpha contract
Purpose	declare a thread procedure

Syntax

```
THREAD identifier([parameter-list])
  LOCAL
    declarations
CODE
  statements
END
```

Parameter	Meaning
identifier	Thread procedure name.
parameter-list	Values copied into the new thread context.

Notes: THREAD declares a routine intended to run outside the UI event loop.

Example

```
THREAD ImportCustomers(FileName : String)
CODE
  ImportFile(FileName)
  POST TO UI MainWindow.ImportFinished
END
```

See also: START THREAD, POST TO UI

START THREAD (launch a thread)

Item	Description
Status	Alpha contract
Purpose	launch a thread

Syntax

```
START THREAD thread-name([arguments]) [AS task-name] [CANCEL cancel-token]
```

Parameter	Meaning
thread-name	THREAD routine.
AS	Optional task handle name.
CANCEL	Optional cancellation token.

Notes: START THREAD schedules background execution and returns a task handle when requested.

Example

```
START THREAD ImportCustomers(Path) AS ImportTask CANCEL ImportCancel
```

See also: THREAD, WAIT

POST TO UI (send work to UI thread)

Item	Description
Status	Alpha contract
Purpose	send work to UI thread

Syntax

```
POST TO UI target.event-name [WITH payload]
```

Parameter	Meaning
target	Window, application, or dispatcher.
event-name	Event to invoke on the UI thread.
payload	Optional message object.

Notes: POST TO UI safely transfers state from a background thread to UI-owned code.

Example

```
POST TO UI MainWindow.RefreshCustomerList WITH Result
```

See also: ON, EVENT

LOCK (protect shared data)

Item	Description
Status	Alpha contract
Purpose	protect shared data

Syntax

```
LOCK lock-name
  statements
END
```

Parameter	Meaning
lock-name	Named lock object.

Notes: LOCK protects a small critical section. Do not perform UI operations while holding a lock.

Example

```
LOCK CacheLock
  CustomerCache[Customer.Id] = Customer
END
```

See also: QUEUE, THREAD

WAIT (wait for task completion)

Item	Description
Status	Alpha contract
Purpose	wait for task completion

Syntax

```
WAIT task-name [TIMEOUT milliseconds]
```

Parameter	Meaning
task-name	Task handle.
TIMEOUT	Optional maximum wait time.

Notes: WAIT blocks the current routine until a task completes. Avoid WAIT on the UI thread unless the timeout is short and the UI remains responsive.

Example

```
WAIT ImportTask TIMEOUT 5000
```

See also: START THREAD

Thread-local and shared data

Data kind	Lifetime	Thread visibility
LOCAL routine data	Routine call	Current thread only.
THREADLOCAL variable	Thread lifetime	Current thread only.
GLOBAL variable	Application lifetime	All threads; synchronization required for mutation.
UI control state	Window lifetime	UI thread only.
Database connection	Connection lifetime	Owning context only unless pooled by driver.

Thread safety rules

1. Do not update controls directly from a worker thread.
2. Pass immutable values or record copies between threads where possible.
3. Keep LOCK sections short and free of UI calls.
4. Use cancellation tokens for long-running work.
5. Treat each database transaction as owned by one logical task context.
6. Write diagnostic logs with the thread id and task name for background errors.
7. Use POST TO UI for progress, completion, and error presentation.

Thread diagnostics

Code	Meaning	Recommended action
SFT0001	UI object accessed from non-UI thread.	Use POST TO UI.
SFT0002	Lock timeout.	Reduce lock scope or investigate deadlock.
SFT0003	Task canceled.	Handle cancellation path and cleanup.
SFT0004	Thread-local data requested before initialization.	Move initialization to thread entry.
SFT0005	Shared variable write without synchronization.	Add LOCK or use a safe collection.

Chapter 6 - Safire Database Drivers

Database runtime overview

Safire database support is dictionary-driven. A dictionary defines tables, fields, keys, relationships, validation, display metadata, browse behavior, update forms, and driver bindings. Runtime drivers translate dictionary operations into physical storage operations while preserving Safire transaction, locking, error, and validation semantics.

Driver categories

Driver category	Purpose
Local file driver	Single-user or small workgroup local data storage.
Server database driver	Networked database engine using a connection and commands.
ODBC driver	Standard connector for installed data sources.
Memory driver	Temporary tables, tests, caches, and generated working sets.
Document driver	Structured import/export sources for business data exchange.

Dictionary source structure

```
DICTIONARY CustomerDictionary
  TABLE Customer DRIVER "Local"
    FIELD Id          : Int AUTONUMBER PRIMARY
    FIELD Name        : String(120) REQUIRED
    FIELD Phone       : String(40)
    FIELD Email       : String(160)
    FIELD Balance     : Decimal(12,2) DEFAULT 0.00
    KEY CustomerPK    ON Id UNIQUE
    KEY CustomerName  ON Name
  END
END
```

Database reference items

DICTIONARY (declare database metadata)

Item	Description
Status	Alpha contract
Purpose	declare database metadata

Syntax

```
DICTIONARY identifier
  table declarations
END
```

Parameter	Meaning
identifier	Dictionary name.

Notes: DICTIONARY is a source artifact. Designers, build tools, reports, browses, and validation use it as the data model source.

Example

```
DICTIONARY Accounts
  TABLE Customer DRIVER "Local"
  END
END
```

See also: TABLE, FIELD

TABLE (declare a persistent table)

Item	Description
Status	Alpha contract
Purpose	declare a persistent table

Syntax

```
TABLE identifier DRIVER string
  FIELD declarations
  KEY declarations
END
```

Parameter	Meaning
identifier	Table name.
DRIVER	Driver binding name.

Notes: TABLE defines stored records and driver binding. A TABLE may map to a physical file, server table, service endpoint, or memory storage depending on driver.

Example

```
TABLE Invoice DRIVER "Server"
  FIELD Id : Int PRIMARY
END
```

See also: FIELD, KEY, DRIVER

FIELD (declare table field)

Item	Description
Status	Alpha contract
Purpose	declare table field

Syntax

```
FIELD identifier : Type [attributes]
```

Parameter	Meaning
identifier	Field name.
Type	Safire type.
attributes	REQUIRED, UNIQUE, DEFAULT, DISPLAY, VALIDATE, etc.

Notes: FIELD defines storage type, validation, display name, default values, and generated browse/update behavior.

Example

```
FIELD Email : String(160) DISPLAY "Email" VALIDATE EmailFormat
```

See also: TABLE, VALIDATE

KEY (declare access path)

Item	Description
Status	Alpha contract
Purpose	declare access path

Syntax

```
KEY identifier ON field-list [UNIQUE] [DESCENDING]
```

Parameter	Meaning
identifier	Key name.
field-list	One or more fields.

Notes: KEY defines an ordered access path for browse, lookup, search, and integrity.

Example

```
KEY CustomerName ON Name
```

See also: TABLE, BROWSE

CONNECT (open database connection)

Item	Description
Status	Alpha contract
Purpose	open database connection

Syntax

```
CONNECT connection-name USING driver-name WITH connection-settings
```

Parameter	Meaning
connection-name	Runtime connection handle.
driver-name	Driver to load.
connection-settings	Project or runtime settings object.

Notes: CONNECT establishes a database runtime connection. Applications normally let the project/runtime open declared connections.

Example

```
CONNECT MainData USING "Server" WITH Config.Data.Main
```

See also: DISCONNECT, TRANSACTION

TRANSACTION (wrap data changes)

Item	Description
Status	Alpha contract
Purpose	wrap data changes

Syntax

```
TRANSACTION connection-name
  statements
COMMIT
[ROLLBACK ON ERROR]
END
```

Parameter	Meaning
connection-name	Connection or unit-of-work context.

Notes: TRANSACTION groups changes into an atomic unit. ROLLBACK ON ERROR is recommended for multi-table updates.

Example

```
TRANSACTION MainData
  Customers.Save(Customer)
  Audit.Write("Customer saved")
COMMIT
ROLLBACK ON ERROR
END
```

See also: CONNECT, LOCK RECORD

LOCK RECORD (reserve a record for update)

Item	Description
Status	Alpha contract
Purpose	reserve a record for update

Syntax

```
LOCK RECORD table-name WHERE condition [TIMEOUT milliseconds]
```

Parameter	Meaning
table-name	Table to lock.
condition	Record selector.

Notes: LOCK RECORD protects update-sensitive business operations. Driver support depends on storage type.

Example

```
LOCK RECORD Customer WHERE Id = CustomerId TIMEOUT 3000
```

See also: TRANSACTION, UNLOCK RECORD

Driver interface contract

Driver operation	Description
Open	Initialize driver, validate settings, and create connection context.
Close	Release resources and flush safe state.
Select	Return records by key, filter, or query plan.
Insert	Create a new record with validation and generated values.
Update	Modify an existing record with locking and validation.
Delete	Remove or mark a record according to table policy.
BeginTransaction	Start a unit of work.
Commit	Persist changes.
Rollback	Undo changes in the current unit.
Describe	Return runtime capabilities for tooling and diagnostics.

Database diagnostics

Code	Meaning	Recommended action
SFD0001	Driver not found.	Check project driver name and packaged runtime files.
SFD0002	Connection failed.	Validate settings and access rights.
SFD0003	Record not found.	Check key values and filters.
SFD0004	Duplicate key.	Handle uniqueness rule in validation or UI.
SFD0005	Lock conflict.	Retry, show conflict, or reduce transaction duration.
SFD0006	Transaction failed.	Review database log and rollback result.
SFD0007	Dictionary mismatch.	Regenerate metadata or migrate storage.

Chapter 7 - Safire Source Code Format

Source file rules

Safire source files should be UTF-8 text. Line endings may be normalized by the repository or editor, but the compiler treats source records as lines. The recommended style places one declaration or statement per line and uses two spaces of indentation for nested blocks.

Module layout

```
! File: src/windows/CustomerList.sf
MODULE CustomerListModule
  USES CustomerTypes

WINDOW CustomerListWindow
  TITLE TEXTKEY "Customer.List.Title" DEFAULT "Customers"
  SIZE 900, 600
  ! controls here
END

PROCEDURE OpenCustomerList
CODE
  OpenWindow(CustomerListWindow)
END
END
```

Source formatting conventions

Convention	Recommended rule
Indentation	Two spaces per nested level.
Keyword casing	Uppercase for language keywords in reference examples.
Identifier casing	UpperCamelCase for application artifacts, lowerCamelCase for short locals if desired.
Line length	Prefer lines under 110 characters.
Continuation	Break long expressions into named local values.
Comments	Use comments to explain business reason, not obvious syntax.
Generated regions	Mark generated blocks clearly and keep them reproducible.
Designer write-back	Do not reorder designer-owned controls by hand unless the change is intentional.

File ownership markers

Generated source may be checked into source control when it is meant to be reviewed and owned. Disposable generated files should remain in build or metadata folders. The header below is recommended for source files that are generated but intentionally reviewed.

```
!!! Safire Source Ownership
!!! Owner: Developer
!!! GeneratedBy: Safire Designer
!!! Regenerate: Safe when source model is unchanged
!!! Purpose: Window source for CustomerListWindow
```

Compiler directives

SECTION (mark a named source section)

Item	Description
Status	Alpha contract
Purpose	mark a named source section

Syntax

```
SECTION "section-name"
  declarations-or-statements
END SECTION
```

Parameter	Meaning
section-name	Stable source section name.

Notes: SECTION is used by tools that insert or update a specific region without rewriting the entire file.

Example

```
SECTION "designer-controls"
  BUTTON SaveButton
  END
END SECTION
```

See also: INCLUDE, DESIGNER

ONCE (prevent repeated include)

Item	Description
Status	Alpha contract
Purpose	prevent repeated include

Syntax

```
INCLUDE "file.sf" ONCE
```

Parameter	Meaning
file.sf	Project-relative source file.

Notes: ONCE prevents duplicate declarations when shared declarations are included from more than one module.

Example

```
INCLUDE "shared/DateRules.sf" ONCE
```

See also: INCLUDE

ASSERT (declare a debug assumption)

Item	Description
Status	Alpha contract
Purpose	declare a debug assumption

Syntax

```
ASSERT condition [MESSAGE string]
```

Parameter	Meaning
condition	Boolean expression.
MESSAGE	Optional message when assertion fails.

Notes: ASSERT is active according to the selected build configuration. It should not replace business validation.

Example

```
ASSERT Customer.Id > 0 MESSAGE "Customer must be loaded"
```

See also: VALIDATE, DEBUG

Reserved words

Reserved	Reserved	Reserved	Reserved	Reserved
APPLICATION	ASSERT	BREAK	CASE	CATCH
CLASS	CODE	CONNECT	CONST	CONTINUE
DEFINE	DICTIONARY	ELSE	ELSIF	END
EVENT	FIELD	FINALLY	FOR	FUNCTION
IF	INCLUDE	KEY	LOCAL	LOCK
LOOP	METHOD	MODULE	NOT	ON
OR	PROCEDURE	RECORD	REPORT	RESOURCE
RETURN	TABLE	TARGET	THEN	THREAD
TRY	USES	VAR	WINDOW	

Chapter 8 - Safire Variable Declaration

Variable declaration overview

Variables are declared with a name, type, optional attributes, and optional initializer. Variables may be local to a routine, fields in a record or class, global to a module, thread-local, or bound to UI controls and table fields.

```
LOCAL
  CustomerId : Int = 0
  CustomerName : String(120)
  Balance : Decimal(12,2) = 0.00
  IsActive : Bool = true
```

Core data types

Type	Purpose	Example
Bool	True or false values.	IsActive : Bool = true
Byte	Small unsigned storage value.	Flags : Byte
Short	Small signed integer.	Level : Short
Int	Default integer.	CustomerId : Int
Long	Large integer.	Sequence : Long
Decimal(p,s)	Business decimal with precision and scale.	Amount : Decimal(12,2)
Number	General numeric value when precision is not fixed.	Ratio : Number
String(n)	Text with optional maximum length.	Name : String(120)
Text	Large text.	Notes : Text
Date	Calendar date.	InvoiceDate : Date
Time	Time of day.	StartTime : Time
DateTime	Date and time.	CreatedOn : DateTime
Guid	Unique identifier.	PublicId : Guid
Blob	Binary data.	DocumentBody : Blob

Declaration attributes

Attribute	Applies to	Meaning
REQUIRED	FIELD, VAR	Value must be present.
DEFAULT	FIELD, VAR	Initial value when none is supplied.
READONLY	VAR, FIELD	Cannot be assigned after initialization.
THREADLOCAL	VAR	One instance per thread.
GLOBAL	VAR	Module or application-level lifetime.
PRIVATE	FIELD, METHOD	Visible only inside owning type or module.
PUBLIC	FIELD, METHOD	Visible to allowed external code.
BIND	VAR, CONTROL	Bind variable to UI or data source.
FORMAT	FIELD, VAR	Display formatting rule.
VALIDATE	FIELD, VAR	Validation rule.

Variable reference items

VAR (declare mutable value)

Item	Description
Status	Alpha contract
Purpose	declare mutable value

Syntax

```
VAR identifier : Type [= expression] [attributes]
```

Parameter	Meaning
identifier	Variable name.
Type	Safire type.
expression	Optional initializer.

Notes: VAR declares a mutable value. In a LOCAL section, the VAR keyword may be omitted when project style allows name-colon-type form.

Example

```
VAR RetryCount : Int = 0
```

See also: CONST, LOCAL

THREADLOCAL (declare per-thread value)

Item	Description
Status	Alpha contract
Purpose	declare per-thread value

Syntax

```
VAR identifier : Type THREADLOCAL [= expression]
```

Parameter	Meaning
identifier	Variable name.
Type	Safire type.

Notes: THREADLOCAL creates one value instance per thread. Use it for task context, last error state, or per-thread caches.

Example

```
VAR CurrentImportFile : String THREADLOCAL
```

See also: THREAD, GLOBAL

ARRAY (declare fixed indexed storage)

Item	Description
Status	Alpha contract
Purpose	declare fixed indexed storage

Syntax

```
identifier : Type DIM count
identifier : ARRAY OF Type SIZE count
```

Parameter	Meaning
Type	Element type.
count	Fixed element count.

Notes: ARRAY storage is indexed from 1 by default unless a project style explicitly sets zero-based indexing.

Example

```
MonthTotals : Decimal(12,2) DIM 12
```

See also: LIST, QUEUE

LIST (declare ordered collection)

Item	Description
Status	Alpha contract
Purpose	declare ordered collection

Syntax

```
identifier : LIST OF Type
```

Parameter	Meaning
Type	Element type.

Notes: LIST is a resizable ordered collection suitable for UI lists, buffers, and intermediate data.

Example

```
SelectedIds : LIST OF Int
```

See also: FOR EACH, QUEUE

QUEUE (declare work queue)

Item	Description
Status	Alpha contract
Purpose	declare work queue

Syntax

```
identifier : QUEUE OF Type [THREADSAFE]
```

Parameter	Meaning
Type	Element type.
THREADSAFE	Enables synchronized enqueue/dequeue.

Notes: QUEUE is intended for ordered processing and thread communication.

Example

```
ImportQueue : QUEUE OF ImportMessage THREADSAFE
```

See also: THREAD, LOCK

Initialization rules

1. Explicit initializers are evaluated when the variable is created.
2. Numeric values default to zero when no initializer is present.
3. Bool values default to false.
4. String values default to an empty string unless nullable policy says otherwise.
5. Record fields are initialized field by field in declaration order.
6. Class instances run field initialization before constructor code.
7. Thread-local values are initialized when the thread context is created.
8. Database fields may apply driver defaults after Safire validation rules run.

Chapter 9 - Safire Windows and Component Structures

Window model overview

Safire windows are source-defined UI artifacts. A window owns controls, layout, data bindings, events, menus, toolbars, status areas, validation display, and designer metadata. The visual designer reads and writes the same source model that the compiler and runtime consume.

Controls are named components. A control name is stable and is used by event handlers, data bindings, validation, and testing. Captions and text may be translated and should not be used as control identity.

Window source example

```
WINDOW CustomerEdit
  TITLE TEXTKEY "Customer.Edit.Title" DEFAULT "Customer"
  SIZE 640, 420
  LAYOUT Grid COLUMNS 2

  LABEL NameLabel
    TEXT TEXTKEY "Customer.Name.Label" DEFAULT "Name"
  END

  TEXTBOX NameEntry
    BIND Customer.Name
    REQUIRED
  END

  BUTTON SaveButton
    TEXT TEXTKEY "Common.Save" DEFAULT "Save"
    ON CLICK
      SaveCustomer()
      CLOSE WINDOW ACCEPTED
    END
  END
END
```

Window reference items

WINDOW (declare a window or form)

Item	Description
Status	Alpha contract
Purpose	declare a window or form

Syntax

```
WINDOW identifier
  TITLE text-expression
  SIZE width, height
  control declarations
END
```

Parameter	Meaning
identifier	Window name.
TITLE	Caption or TEXTKEY.
SIZE	Designed width and height.

Notes: WINDOW is a top-level visual artifact. It may be opened modally or modelessly depending on call site.

Example

```
WINDOW MainWindow
  TITLE "Main"
  SIZE 900, 600
END
```

See also: CONTROL, OPEN WINDOW

DIALOG (declare modal interaction)

Item	Description
Status	Alpha contract
Purpose	declare modal interaction

Syntax

```
DIALOG identifier
  TITLE text-expression
  SIZE width, height
  control declarations
END
```

Parameter	Meaning
identifier	Dialog name.

Notes: DIALOG is a window intended to return an explicit result such as Accepted or Cancelled.

Example

```
DIALOG CustomerLookup
  TITLE "Select customer"
  SIZE 700, 500
END
```

See also: WINDOW, CLOSE WINDOW

CONTROL (base control declaration)

Item	Description
Status	Alpha contract
Purpose	base control declaration

Syntax

```
CONTROL identifier TYPE control-type
  properties
  events
END
```

Parameter	Meaning
identifier	Control name.
control-type	Button, TextBox, Browse, etc.

Notes: CONTROL is the generic form. Most source uses specific control keywords for readability.

Example

```
CONTROL SaveButton TYPE Button
  TEXT "Save"
END
```

See also: BUTTON, TEXTBOX

ON (declare event handler)

Item	Description
Status	Alpha contract
Purpose	declare event handler

Syntax

```
ON event-name
  statements
```

```
END
```

Parameter	Meaning
event-name	CLICK, CHANGE, OPEN, CLOSE, SELECT, TIMER, etc.

Notes: ON attaches source code to a component event. Event code belongs to the component or window source unless factored into a procedure.

Example

```
ON CLICK
  SaveCustomer()
END
```

See also: EVENT, POST TO UI

BIND (bind control to data)

Item	Description
Status	Alpha contract
Purpose	bind control to data

Syntax

```
BIND expression
```

Parameter	Meaning
expression	Variable, record field, table field, or view field.

Notes: BIND keeps control value and source value synchronized according to binding mode.

Example

```
TEXTBOX EmailEntry
  BIND Customer.Email
END
```

See also: FIELD, VALIDATE

Standard controls

Control	Purpose	Key events
LABEL	Static text.	
TEXTBOX	Text input.	CHANGE, VALIDATE, ENTER
NUMBERBOX	Numeric input.	CHANGE, VALIDATE
DATEBOX	Date input.	CHANGE, VALIDATE
CHECKBOX	Boolean input.	CLICK, CHANGE
COMBOBOX	Pick from list.	SELECT, CHANGE
LISTBOX	List selection.	SELECT, DOUBLECLICK
BROWSE	Tabular data browsing.	SELECT, ACCEPT, SORT, FILTER
BUTTON	Command button.	CLICK
IMAGE	Display image.	CLICK
TAB	Tabbed container.	SELECT
PANEL	Layout container.	OPEN, RESIZE
MENU	Window menu.	SELECT
TOOLBAR	Command surface.	CLICK
STATUSBAR	Status messages.	

Layout structures

Layout	Description
Absolute	Fixed coordinates. Useful for imported screens and precise forms.
Grid	Rows and columns with resizing rules. Recommended for data-entry forms.
Stack	Vertical or horizontal stacking with spacing.
Dock	Top, bottom, left, right, and fill regions.
Flow	Controls flow according to available space.
Splitter	Resizable panes.
Notebook	Tabbed pages.

Window lifecycle events

Event	When it runs	Common use
CREATE	Runtime object is being created.	Initialize nonvisual state.
OPEN	Window is opening.	Load data, set defaults.
ACTIVATE	Window becomes active.	Refresh commands.
RESIZE	Window size changes.	Adjust custom drawing.
VALIDATE	User attempts to accept data.	Business validation.
ACCEPT	Window accepted.	Save data.
CANCEL	Window cancelled.	Undo or ignore changes.
CLOSE	Window is closing.	Release resources.

Chapter 10 - Safire Reports and Component Structures

Report model overview

Safire reports are source-defined artifacts with bands, fields, expressions, totals, groups, parameters, page setup, preview, print, and export behavior. Reports may be designed visually, but the report source remains the developer-owned definition.

Report source example

```
REPORT CustomerStatement
  TITLE TEXTKEY "Report.CustomerStatement.Title" DEFAULT "Customer Statement"
  PAPER A4 PORTRAIT
  MARGINS 12, 12, 12, 12
  PARAMETER CustomerId : Int REQUIRED

  DATASET CustomerData FROM Customer WHERE Id = CustomerId
  DATASET InvoiceData FROM Invoice WHERE CustomerId = CustomerId ORDER BY InvoiceDate

  BAND PAGEHEADER HEIGHT 24
    TEXT ReportTitle VALUE TEXTKEY "Report.CustomerStatement.Title" DEFAULT "Customer Statement"
  END

  BAND DETAIL HEIGHT 18 DATASET InvoiceData
    FIELD InvoiceDate VALUE InvoiceData.InvoiceDate FORMAT "Date.Short"
    FIELD InvoiceTotal VALUE InvoiceData.Total FORMAT "Money.Standard"
  END
END
```

Report reference items

REPORT (declare report artifact)

Item	Description
Status	Alpha contract
Purpose	declare report artifact

Syntax

```
REPORT identifier
  TITLE text-expression
  PAPER paper-name orientation
  report-components
END
```

Parameter	Meaning
identifier	Report name.
TITLE	Report title.
PAPER	Paper and orientation.

Notes: REPORT is a top-level printable and exportable artifact.

Example

```
REPORT InvoicePrint
  TITLE "Invoice"
  PAPER A4 PORTRAIT
END
```

See also: BAND, DATASET

BAND (declare report section)

Item	Description
Status	Alpha contract
Purpose	declare report section

Syntax

```
BAND band-type [HEIGHT value] [DATASET name]
  report controls
END
```

Parameter	Meaning
band-type	PAGEHEADER, HEADER, DETAIL, FOOTER, PAGEFOOTER, GROUPHEADER, GROUPFOOTER.
HEIGHT	Band height.

Notes: BAND groups report output controls and controls repetition.

Example

```
BAND DETAIL HEIGHT 18 DATASET InvoiceLines
END
```

See also: REPORT, FIELD

DATASET (declare report data source)

Item	Description
Status	Alpha contract
Purpose	declare report data source

Syntax

```
DATASET identifier FROM table-or-query [WHERE condition] [ORDER BY field-list]
```

Parameter	Meaning
identifier	Dataset name.
FROM	Table, view, or query.

Notes: DATASET supplies rows to detail or group bands.

Example

```
DATASET Lines FROM InvoiceLine WHERE InvoiceId = Parameter.InvoiceId
```

See also: TABLE, QUERY

PARAMETER (declare report parameter)

Item	Description
Status	Alpha contract
Purpose	declare report parameter

Syntax

```
PARAMETER identifier : Type [REQUIRED] [DEFAULT expression]
```

Parameter	Meaning
identifier	Parameter name.
Type	Safire type.

Notes: PARAMETER is supplied by the calling window, procedure, or report preview dialog.

Example

```
PARAMETER FromDate : Date REQUIRED
```

See also: REPORT, DATASET

TOTAL (declare report total)

Item	Description
Status	Alpha contract
Purpose	declare report total

Syntax

```
TOTAL identifier = aggregate(expression) [RESET band-or-group]
```

Parameter	Meaning
identifier	Total name.
aggregate	SUM, COUNT, MIN, MAX, AVG.

Notes: TOTAL calculates summary values for display in footers.

Example

```
TOTAL InvoiceTotal = SUM(InvoiceLines.LineTotal) RESET GROUP CustomerId
```

See also: GROUP, BAND

Report controls

Control	Purpose
TEXT	Fixed text, translated text, or expression text.
FIELD	Bound dataset value.
LINE	Line drawing.
BOX	Rectangular border or background.
IMAGE	Logo, product image, signature, or embedded picture.
BARCODE	Encoded value for scanning when enabled by runtime.
SUBREPORT	Nested report region.
PAGEINFO	Page number, page count, date, and runtime values.

Report output lifecycle

1. Prepare parameters and validate required inputs.
2. Open datasets and execute query plans.
3. Run BEFORE PRINT report event.
4. Render page header and group headers.
5. Render detail bands for each dataset row.
6. Compute totals and render footers.
7. Paginate using paper, margins, and keep-together rules.
8. Preview, print, export, or write PDF according to caller request.
9. Close datasets and release runtime resources.

Report diagnostics

Code	Meaning	Recommended action
SFRP0001	Report source not found.	Check project report path.
SFRP0002	Dataset failed.	Review dataset source and parameters.
SFRP0003	Field expression failed.	Check field name, type, and formatting.
SFRP0004	Band overflow.	Reduce content or enable grow/split.
SFRP0005	Export failed.	Check output path and runtime export support.
SFRP0006	Missing parameter.	Supply required parameter before preview or print.

Chapter 11 - Error Codes and Debugging

Diagnostic model

Safire diagnostics are designed to be explicit, searchable, and actionable. Compiler, build, runtime, database, report, internationalisation, and thread errors use prefixes that indicate their subsystem. Each diagnostic should include a code, severity, message, source location when available, and suggested action.

Diagnostic record format

```
{
  "code": "SFC0201",
  "severity": "Error",
  "message": "Unknown identifier CustomerNam",
  "file": "src/windows/CustomerEdit.sf",
  "line": 42,
  "column": 11,
  "suggestion": "Did you mean CustomerName?"
}
```

Error code prefixes

Prefix	Subsystem
SFC	Compiler and semantic checker.
SFB	Build and project system.
SFR	Runtime and VM.
SFD	Database drivers and dictionary runtime.
SFW	Windows, controls, layout, and UI runtime.
SFRP	Reports, preview, print, and export.
SFT	Threading and task dispatch.
SFI	Internationalisation and resource catalogs.
SFA	AI assistant audit and apply workflow.
SFP	Packaging, release, and deployment.

Compiler diagnostics

Code	Severity	Meaning	Action
SFC0001	Error	Source file not found.	Check project path and sourceRoots.
SFC0100	Error	Parse error.	Check syntax near the reported line.
SFC0200	Error	Unknown symbol.	Check spelling, module imports, and declaration order.
SFC0201	Error	Ambiguous symbol.	Qualify the name with module or owner.
SFC0300	Error	Type mismatch.	Convert explicitly or fix declaration type.
SFC0400	Error	Invalid assignment.	Confirm variable mutability and target type.
SFC0500	Warning	Unreachable code.	Remove dead code or correct control flow.
SFC0600	Warning	Unused declaration.	Remove or mark as intentionally exported.

Build diagnostics

Code	Severity	Meaning	Action
SFB0001	Error	Project manifest invalid.	Validate .sfproj JSON and required fields.
SFB0002	Error	No active project.	Select active project in .sfsol.
SFB0003	Error	Output path unavailable.	Check permissions and clean build folder.
SFB0004	Error	Runtime dependency missing.	Refresh runtime package and rebuild.
SFB0005	Warning	Derived metadata stale.	Clean metadata and rebuild.
SFB0006	Error	Native target failed.	Read build log and verify target settings.

Runtime diagnostics

Code	Severity	Meaning	Action
SFR0001	Error	Bytecode file missing.	Rebuild or correct launch path.
SFR0002	Error	Runtime version mismatch.	Package matching runtime with application.
SFR0100	Error	Null value access.	Validate object before member access.
SFR0200	Error	Index out of range.	Check list bounds.
SFR0300	Error	Unhandled exception.	Add TRY/CATCH around expected failures.
SFR0400	Warning	Slow operation on UI thread.	Move long work to THREAD.

Debugging workflow

1. Run a semantic check before build when changing source structure.
2. Read the first error first. Later errors may be caused by the first one.
3. Use source locations from diagnostics rather than searching generated files.
4. Reproduce runtime failures with the same project configuration and resources.
5. Inspect metadata/build/last-build.json for the exact output paths used by tools.
6. Create a support bundle when the problem depends on runtime files, project manifests, or packaged output.
7. For UI issues, verify control names, layout rules, data binding, and event handlers.
8. For database issues, check dictionary metadata, driver settings, transaction state, and locks.

Debugging reference items

DEBUG (enable debug-only behavior)

Item	Description
Status	Alpha contract
Purpose	enable debug-only behavior

Syntax

```
DEBUG
  statements
END
```

Notes: DEBUG blocks run only when debug configuration includes debug execution. Avoid business behavior that exists only in DEBUG blocks.

Example

```
DEBUG
  Log("Loaded customer " + Customer.Id)
END
```

See also: ASSERT, LOG

LOG (write diagnostic message)

Item	Description
Status	Alpha contract
Purpose	write diagnostic message

Syntax

```
LOG(message [, category] [, level])
```

Parameter	Meaning
message	Text to write.
category	Optional subsystem or application category.
level	Trace, Info, Warning, Error.

Notes: LOG writes to the configured diagnostic sink and should include useful context.

Example

```
LOG("Saving customer " + Customer.Id, "Customer", "Info")
```

See also: TRACE, SUPPORT BUNDLE

SUPPORT BUNDLE (collect diagnostics)

Item	Description
Status	Alpha contract
Purpose	collect diagnostics

Syntax

```
SafireBuild.exe support-bundle project.sfproj --out path
```

Parameter	Meaning
project.sfproj	Project manifest.
path	Output archive path.

Notes: A support bundle collects project manifest, logs, build summaries, environment summaries, and selected metadata. It must not include private data unless the developer explicitly allows it.

Example

```
SafireBuild.exe support-bundle CustomerManager.sfproj --out support/customer.zip
```

See also: LOG, BUILD

Chapter 12 - Runtime Services

Safire runtime services provide common business application capabilities: configuration, files, timers, dates, text, resources, messaging, validation, dialogs, printing, export, application state, and controlled external interaction. Runtime services should be accessed through Safire APIs so that applications remain buildable, diagnosable, and portable within the Safire ecosystem.

Core runtime services

Service	Purpose
Application	Startup, shutdown, command-line arguments, current user, and runtime paths.
Configuration	Settings, environment-specific values, and connection definitions.
FileSystem	Project-approved file and folder operations.
Resource	Localized text, images, and help resources.
Dialog	Message boxes, file dialogs, confirmation prompts, and validation display.
Timer	Scheduled UI and background callbacks.
Validation	Shared business validation and user feedback.
Report	Preview, print, and export support.
Data	Connections, transactions, cursors, and repository access.
Security	Current user, roles, permissions, and audit metadata.

Service access pattern

```
PROCEDURE SaveCustomer(Customer : CustomerRecord) RETURNS Bool
CODE
  IF NOT Validate(Customer)
    Dialog.ShowError(TEXTKEY "Customer.Invalid" DEFAULT "Customer is invalid")
    RETURN false
  END

  Data.Customers.Save(Customer)
  Log("Customer saved", "Data", "Info")
  RETURN true
END
```

Runtime path rules

Path	Use
ApplicationPath	Folder containing the running application.
DataPath	Writable application data.
ConfigPath	Configuration files.
LogPath	Runtime logs.
TempPath	Temporary files owned by the current run.
ResourcePath	Packaged resources.
ReportPath	Report templates and export output when configured.

Chapter 13 - Deployment and Packaging

A Safire deployed application should not require the developer IDE. The packaged output contains the application artifact, runtime components when needed, database drivers, report/export runtime support, configuration, resource catalogs, and support diagnostics appropriate for the target environment.

Package manifest

```
{
  "packageVersion": 1,
  "application": "CustomerManager",
  "configuration": "Release",
  "artifacts": [
    { "path": "CustomerManager.exe", "role": "application" },
    { "path": "runtime/SafireRun.exe", "role": "runtime" },
    { "path": "resources/text.en.sfres", "role": "resource" }
  ],
  "createdBy": "SafireBuild"
}
```

Package contents

Item	Required	Purpose
Application executable	Yes for native target	Main runtime artifact.
Bytecode artifact	Yes for VM target	Compiled Safire program.
Runtime launcher	When VM target is used	Runs bytecode and runtime metadata.
Database drivers	When data access is used	Database runtime connectivity.
Report/export files	When reports are used	Preview, print, and export support.
Resource catalogs	When localized or external text is used	User-facing text and resources.
Configuration	As needed	Runtime settings and environment binding.
Support tools	Recommended	Collect diagnostics without the IDE.

Release checklist

1. Build the project in Release configuration.
2. Run package manifest verification.
3. Verify required runtime files are present.
4. Run the application from the package folder, not from the development folder.
5. Test database connectivity with package settings.
6. Open every packaged report that is part of the acceptance path.
7. Create a support bundle from the package folder.
8. Copy the package to a clean folder and repeat a smoke run.
9. Record package manifest, build log, and acceptance notes.

Chapter 14 - Project-Aware AI Workflow

Safire AI assistance is project-aware and reviewable. The AI workflow is not a replacement for compiler diagnostics, source review, or build verification. AI proposals should be based on Safire project context and applied through controlled diff and audit steps.

AI workflow stages

Stage	Description
Question or request	Developer states the task in application terms.
Context build	Safire collects relevant project source, diagnostics, and metadata.
Proposal	AI suggests explanation, source changes, or a plan.
Human review	Developer reviews the proposal and decides whether to continue.
Diff	Tooling presents exact source changes.
Approved apply	Changes are written only after approval.
Rollback point	Recoverable state is retained according to project policy.
Audit log	Request, proposal, diff, apply result, and build result are recorded.
Build/test proof	Compiler and build output confirm whether the change is real.

Safe AI rules

1. AI must not become the only place where application design exists.
2. AI-generated changes must be reviewable before being applied.
3. AI context should be limited to relevant project source and diagnostics.
4. Generated patches should change source artifacts, not hidden caches.
5. After applying a change, run semantic check or build proof.
6. Audit logs should identify files changed and commands executed.
7. Rollback should restore source to a known previous state when requested.

AI artifact folders

Folder	Purpose
metadata/ai/context	Prepared project context snapshots.
metadata/ai/proposals	AI text proposals.
metadata/ai/diffs	Reviewable patch files.
metadata/ai/audit	Apply logs and decisions.
metadata/ai/results	Build/test proof after AI changes.

Chapter 15 - Developer Style, Review, and Source Ownership

Safire projects are expected to live for years. Style, review, and source ownership rules should optimize for maintenance, diagnostics, safe tooling, and calm handover between developers.

Source ownership principles

1. The developer owns project source, window source, report source, dictionary source, class source, and procedure source.
2. Build output and generated caches are disposable unless explicitly promoted to source.
3. Designers must write back to canonical source rather than private-only storage.
4. Every generated artifact should either be reproducible or clearly marked as developer-owned.
5. Project files should use relative paths so that the project can move between folders and machines.
6. A reviewer should be able to understand important application behavior from source and manifests.

Review checklist

Area	Questions
Language	Are declarations clear, typed, and in the right module?
Windows	Are control names stable and event handlers readable?
Reports	Are datasets, bands, parameters, and totals obvious?
Database	Do table fields, keys, validation, and transactions match the business rule?
Internationalisation	Are user-facing strings resource-backed?
Build	Does the project build from .sfproj without hidden state?
Threading	Does background work avoid direct UI mutation?
Diagnostics	Do errors have enough context for future support?

Version-control checkpoint policy

A checkpoint is useful after a successful code implementation and verified build path. It should not be created silently. The developer should explicitly approve checkpoint creation so that the project history records intentional stable points rather than speculative intermediate states.

Appendix A - Quick Reference

Common commands

```
SafireBuild.exe build Project.sfproj
SafireBuild.exe clean Project.sfproj
SafireBuild.exe package Project.sfproj --configuration Release
SafireBuild.exe support-bundle Project.sfproj --out support/project-support.zip
safirec.exe check src/App.sf --project Project.sfproj
SafireRun.exe build/Project.sfbc
```

Common file extensions

Extension	Meaning
.sf	Safire source.
.sfproj	Project manifest.
.sfsol	Solution/workspace manifest.
.sfdict	Dictionary source.
.sfres	Resource catalog.
.sfwin	Window source.
.sfreport	Report source.
.sfbc	Compiled bytecode artifact.
.sflog	Safire diagnostic log where configured.

Common event names

Event	Typical owner
CREATE	Window/control.
OPEN	Window/report.
CLICK	Button, image, menu item, toolbar item.
CHANGE	Input control.
SELECT	Browse, list, tab, menu.
VALIDATE	Window, control, table field.
ACCEPT	Dialog/window.
CANCEL	Dialog/window.
CLOSE	Window/report.
TIMER	Timer component.

Appendix B - Glossary

Term	Meaning
Application	Runnable Safire project artifact with startup behavior.
Artifact	Developer-owned or generated file used by the project.
Bytecode	Compiled Safire runtime artifact.
Dictionary	Source-defined data model for tables, fields, keys, validation, and driver binding.
Driver	Runtime component that connects dictionary operations to storage.
Manifest	Machine-readable project or package description.
Metadata	Generated tool-readable information derived from source.
Report band	A repeating or fixed report section.

Term	Meaning
Source-first	The rule that source artifacts are the authority for application design.
Window	Source-defined UI artifact containing controls, layout, events, and bindings.

Appendix C - Alpha Implementation Notes

This manual documents the developer-facing Safire1 Version 1 Alpha contract. Some APIs and tools may still be tightened as the product hardens. Where a topic is marked Alpha contract, developers should treat it as the intended surface for source, documentation, testing, and review. Where a topic is marked Reserved, use it only after the current toolchain confirms support.

The most important invariant is that application truth remains in source: project manifests, application declarations, windows, reports, dictionaries, classes, procedures, resources, and build settings. Derived output can be regenerated and should not become the only authoritative application definition.