SafireIDE AI Assistant Pad

Business Case and User Guide

For mature business-application developers using SafireIDE and the Safire language

Version 1.0 Developer Preview Reference
Prepared from the current Safire Version 1 project progress baseline
2026-05-30

Table of Contents

1. Executive summary

Safire is being shaped as a Windows-first, source-first business application development system for experienced developers who build, maintain, modernize, and extend long-lived operational software.

The SafireIDE AI Assistant Pad is the product feature that places AI inside the business application development workflow. It is not intended to be a loose coding chatbot. It is intended to understand Safire projects as business systems made of windows, controls, events, tables, queries, reports, procedures, classes, validation rules, build output, deployment packages, and support evidence.

The business value of the AI Pad is speed with control. It should help the developer find source, explain behavior, propose improvements, generate business UI and report source, diagnose build failures, and prepare safe patches, while preserving human review, diff visibility, backups, audit history, and build/test proof.

Version 1 positioning

At the current Version 1 progress baseline, Safire is no longer only a compiler experiment or design concept. It has moved into a usable first-product path with SafireIDE, source-backed windows and reports, compiler/CLI regression protection, runtime smoke tests, database runtime, browse/update generation, report runtime, sample business application packaging, support bundle direction, licensing boundaries, and a first usable V1 package chain.

Primary audience

This guide is written for mature business-software developers and small teams. The target user is a developer who understands customer data, invoices, stock, payroll, accounting-style workflows, reports, imports, exports, devices, support issues, and deployment responsibility. Safire AI should respect that developer rather than replacing their judgement.

Main promise

> **NOTE:** SafireIDE AI assistance must remain source-aware, reviewable, reversible, auditable, and useful for real business applications.

2. Current Safire Version 1 progress baseline

This document reflects the current Safire project progress as a Version 1 product baseline. The details below are written from a user and business perspective, not as proof-script instructions.

Product areas now represented

Area	Current Version 1 direction
SafireIDE shell	Project tree, source editor, visual design surfaces, property/event/data-binding panels, build/run/output loop, support and diagnostics workflow, and dockable AI Assistant Pad direction.
Source-first model	Windows, reports, controls, events, dictionary/table declarations, queries, procedures, classes, project settings, build settings, and deployment metadata are treated as developer-owned source concepts.
Business UI	Source-backed form/window declarations, control properties, event handlers, data binding, browse/update patterns, preview paths, property-grid edit preview, approved write-back, backup, conflict detection, and review history.
Reports	Report declarations, bands, fields, expressions, grouping, totals, page setup, preview, print/PDF/export direction, visual report designer, validation, diff visibility, and source write-back.
Compiler and CLI	Command-line checking/building remains an important non-IDE path. Recent progress includes compiler/CLI build regression protection and runtime/CLI independence from licensed IDE gating.
Runtime	Runtime smoke-test application, database runtime minimal V1, browse/update generator minimal V1, report runtime minimal V1, and sample business application V1.
Deployment	End-user deployment package V1, developer install package direction, fresh machine install and first launch proof, product health regression runner, release-candidate freeze, and first usable V1 package.
AI and Local Agent	Source indexing, context building, model calls, audit trail, patch proposals, diff review, approved apply with backup, build/test runner, failure analysis, hotfix proposal loop, review history, and restore from backup workflow.
Licensing	Licensed SafireIDE productivity boundary, licensed menu/toolbar command gating, trial/grace/expired state behavior, license diagnostics, support bundle direction, and runtime/CLI independence.

AI-specific progress chain

The AI work has progressed from a local agent concept into a controlled developer workflow. SafireAgentHost is the Safire-controlled authority for source access, search, patch proposal, approved apply, backup, and build/test execution. A model backend can provide reasoning and text proposals, but it does not directly own the project folder.

- Live local-model session and audit trail path established.
- Source search index and ranking improved for useful code results.
- Source context builder created to package relevant project evidence.
- Patch proposal package and diff review workflow established.
- Approved patch apply path created with backup evidence.
- SafireIDE AI panel direction connected to the controlled agent workflow.
- Approved build/test runner and milestone failure analyzer direction established.
- Review history, restore preview, approved restore, and audit entries established for write-back workflows.

User-facing meaning

For the end user of SafireIDE, the practical meaning is simple: the AI Pad should be able to help with the real project, but it should not become an uncontrolled editor. It should explain, propose, show evidence, ask for approval where necessary, apply through Safire-controlled services, and preserve a trail of what changed and why.

3. Business case for the AI Assistant Pad

The business problem

Long-lived business applications are valuable and difficult to replace. They often include hundreds of forms, reports, tables, queries, business rules, imports, exports, device integrations, and deployment habits. The developer responsible for them must maintain correctness while responding quickly to customers.

- Business rules are spread across screens, procedures, reports, validation, database tables, and deployment scripts.
- Existing systems accumulate technical decisions that are not always documented.
- Finding the right source location can take longer than making the actual change.
- Generic AI coding tools are risky when they do not understand the project model or the source-first rule.
- Generated code and designer metadata can become dangerous if they diverge from source.
- Small teams need better support, diagnostics, and release evidence without heavy enterprise process.

The Safire answer

Safire places AI inside a RAD-aware, source-backed development environment. The AI Pad is designed to understand business application artifacts instead of only isolated code files. It should know the difference between a window, a browse, a table, a report band, a query, an event handler, a validation rule, and a deployment package.

Return on investment drivers

Driver	Business benefit
Faster source discovery	Developers spend less time finding the event, report, table, or procedure that needs attention.
Safer changes	Changes are proposed as reviewable diffs and applied only through approved workflows.
Lower onboarding cost	New developers can ask the AI Pad to explain project structure and business flows.
Better consistency	Browse/update windows, validation, reports, data binding, and error handling can follow project conventions.
Better support response	Logs, diagnostics, and support bundles can be interpreted inside the same development environment.
Commercial defensibility	The licensed IDE provides productivity and governance beyond plain text editing or command-line building.
Reduced lock-in fear	The source remains readable and buildable outside the IDE, while the licensed IDE adds professional productivity.

Target customer profile

The best early customer is a developer, consultant, or small team maintaining operational software for real businesses. They care about stability, data, reports, deployment, customer support, and predictable maintenance more than novelty.

4. Commercial and licensing position

Licensing is important to Safire because the professional value is not only the compiler. The licensed product is the complete SafireIDE productivity environment: visual designers, specialist editors, AI Pad, review history, patch workflow, diagnostics, build/run integration, templates, wizards, and support tooling.

Product boundary

Capability	Recommended licensing position
Safire source language	Developer-owned project source. It should remain readable and maintainable.
Compiler and CLI path	Available enough for checking/building projects without forcing all maintenance through the licensed IDE.
Runtime and end-user packages	Independent from the licensed IDE. End users should not need SafireIDE installed to run a deployed application.
SafireIDE	Licensed professional productivity product.
AI Assistant Pad	Licensed IDE feature because it provides project-aware productivity, patch review, audit, diagnostics, and integration.
Advanced visual designers	Licensed IDE productivity features.
Support bundle and license diagnostics	Part of the commercial support and licensing workflow.

License lifecycle behavior

The current Safire progress includes licensed IDE command gating, license lifecycle states, license diagnostics, and runtime/CLI independence. This supports a commercial model where an expired or invalid IDE license can restrict professional IDE features without breaking already deployed end-user applications.

Suggested public message

NOTE: Safire keeps your application source under your control, while the licensed SafireIDE gives you the professional visual design, AI assistance, diagnostics, and productivity layer.

Avoiding the wrong message

CAUTION: Do not position the AI Pad as a magic auto-programmer that silently changes business systems. Position it as a controlled assistant for experienced developers.

5. SafireIDE AI Assistant Pad overview

The AI Assistant Pad is a dockable panel inside SafireIDE. It should remain close to the developer workspace, commonly docked on the left or right side. It is part of the IDE workbench, not a separate generic chat window.

What the AI Pad does

- Answers questions about the current Safire project.
- Searches and explains source files and project artifacts.
- Summarizes windows, controls, events, data bindings, tables, queries, procedures, classes, and reports.
- Helps write Safire source in the approved language style.
- Suggests source changes as reviewable proposals.
- Shows diffs before applying changes.
- Applies only approved changes through Safire-controlled tooling.
- Creates backups and audit evidence for applied changes.
- Runs approved build/test/proof steps and summarizes results.
- Helps diagnose compiler, runtime, report, deployment, and licensing errors.
- Helps write documentation, release notes, support notes, and migration plans from project evidence.

What the AI Pad should not do

- Silently rewrite source files.
- Bypass review, diff, approval, backup, or audit rules for project changes.
- Treat generated caches or runtime plans as the primary source of truth when source files exist.
- Invent project facts when the source or logs do not support them.
- Hide build failures behind vague summaries.
- Require end-user deployment packages to include SafireIDE.

Suggested pad layout

Pad area	Purpose
Project context bar	Shows active project, branch/build configuration, and context status.
Conversation area	Developer questions, assistant answers, summaries, and next-step guidance.
Context panel	Files, windows, reports, tables, logs, or diagnostics selected for the current answer.
Proposal panel	Patch package, affected files, risk notes, and diff preview.
Action buttons	Search, build context, propose patch, show diff, apply approved, run check/build, create support bundle.
History panel	Prior proposals, applied changes, backups, audit entries, and restore options.

6. Architecture and trust model

Core control rule

Safire controls the project. The AI model backend may reason and propose. SafireIDE and SafireAgentHost own source access, patching, backups, approval boundaries, build/test execution, audit trails, and diagnostics.

```
Developer request
 -> SafireIDE AI Assistant Pad
 -> SafireAgentHost builds source context
 -> Model backend returns answer or proposal
 -> SafireIDE shows response, evidence, and diff
 -> Developer approves, edits, or rejects
 -> Safire-controlled apply creates backup and audit entry
 -> SafireIDE runs approved check/build/test
 -> Result is shown in output, review history, and support evidence
```

Source-first safety

The AI Pad must treat Safire source as the durable truth. Generated runtime plans, indexes, caches, previews, build output, proof logs, and support bundles are useful evidence, but they are not the main editable application design.

Audit evidence

- Original developer request.
- Project context and files used.
- AI answer or proposal summary.
- Patch package and diff.
- Approval or rejection decision.
- Backup path for changed files.
- Build/test command run after the change.
- Build/test result and log path.
- Optional restore or rollback entry.

Local model and external model options

Safire can support a local model backend where privacy, offline control, or customer policy matters. It can also support hosted model paths where allowed. In both cases, Safire-controlled tooling should remain the authority over disk, source, patching, and build actions.

7. User guide: first launch and configuration

Open the AI Assistant Pad

1. Open SafireIDE.
2. Open or create a Safire project.
3. Choose View > AI Assistant Pad, or use the toolbar button if your license enables it.
4. Dock the pad on the left or right side of the IDE workspace.
5. Confirm that the active project shown in the pad is the intended project.

Configure the model endpoint

A local or hosted model backend should be configured through SafireIDE settings or a controlled environment/configuration mechanism. The developer should not need to place secrets directly into project source files.

Setting	Meaning
Endpoint URL	The model service endpoint used by SafireAgentHost.
Authentication token	API token or key, preferably supplied through a protected environment variable or secure local configuration.
Default model	The model name used for ordinary project questions.
Timeout	Maximum time allowed for a request before Safire returns control.
Context limit	Maximum project context packaged into a single request.
Audit path	Location where AI session logs and proposal evidence are written.

Check readiness

6. Ask the AI Pad to summarize the active project.
7. Ask it to list the main windows and reports.
8. Ask it to find a known procedure or table.
9. Run a source search and confirm the result file list is sensible.
10. Run a check/build command from the IDE and confirm output is captured.

NOTE: If the AI Pad cannot identify the active project or source root, do not apply changes until the project context problem is fixed.

8. User guide: asking questions and building context

The AI Pad is most useful when the developer asks project-specific questions. Short generic prompts are less useful than prompts that mention the business artifact, source object, error, or change goal.

Good question patterns

Developer asks	AI Pad should do
Explain WIN_CustomerList.	Find the window source, summarize controls, data source, events, and related procedures.
Where is customer validation done?	Search validation blocks, save procedures, customer service classes, and edit window events.
Why does this report not show inactive customers?	Find report data source, query filter, parameters, and relevant report source.
What changed in the last source write-back?	Open review history, show audit entry, affected files, and diff summary.
Why did the build fail?	Read compiler/build output, map diagnostics to source files, and suggest a safe fix.

Build context before asking for changes

For a change request, the AI Pad should first build context from relevant source files. A good workflow is: search, summarize, propose, diff, approve.

```
Ask question -> Search source -> Build context -> Explain current behavior -> Propose change -> Show diff -> Approve apply -> Build/check
```

Example prompt

```
Find the source for the customer browse window.
Explain the table, search field, buttons, and double-click event.
Do not change files yet.
```

Example response standard

A good answer should name the source files used, summarize the relevant business behavior, point out any uncertainty, and offer a next step without applying changes automatically.

9. User guide: source search, explanation, and review

Source search

The AI Pad should expose search as a first-class action. Search results should prioritize user-owned source files over generated output unless the user is specifically diagnosing generated artifacts.

Search target	Typical sources
Window or form	windows/*.sfwin, related procedures, related queries, validation blocks.
Report	reports/*.sfrpt, queries, dictionary tables, report runtime logs.
Database table	dictionary/*.sfdict, queries, browse/update generators, validation.
Procedure or function	procedures/*.sf, classes/*.sf, event handlers.
Build error	compiler output, source file and line, build logs, current configuration.
Deployment problem	package manifest, runtime files, support bundle, diagnostics output.

Explanation checklist

- What artifact is being explained?
- Which source files were used?
- What controls, events, tables, reports, or procedures are involved?
- What business rule is being implemented?
- What generated evidence or logs were considered?
- What is certain and what needs developer confirmation?

Review without changing

Many AI Pad sessions should end with an explanation, not a code change. For mature business developers, understanding is often more valuable than automatic editing.

10. User guide: proposals, diffs, approved apply, and rollback

Proposal workflow

When the developer asks the AI Pad to change a project, the AI Pad should prepare a proposal package rather than immediately editing source files.

11. Developer asks for a change.
12. AI Pad searches and builds context.
13. AI Pad explains current behavior and intended change.
14. AI Pad creates a patch proposal package.
15. SafireIDE displays affected files and diff preview.
16. Developer approves, edits, or rejects the proposal.
17. Safire-controlled apply creates backup and audit entry.
18. SafireIDE runs approved check/build/test.
19. Result is stored in review history.

Diff review questions

- Does the diff touch only the expected source files?
- Does it modify user-owned source rather than generated cache files?
- Are event handlers still short and readable?
- Are business rules placed in procedures, classes, or validation blocks where appropriate?
- Are reports, windows, queries, and tables still source-backed?
- Does the change compile or at least pass the relevant check?

Backup and restore

Every approved apply should create a backup of changed files. Review history should allow the developer to inspect what was applied and restore from backup where appropriate. Restore actions should themselves be auditable.

CAUTION: Do not use AI apply for large structural refactors until the proposal can be reviewed in small, understandable patches.

11. User guide: build, run, diagnostics, and support bundles

Build and run loop

The AI Pad should connect to the IDE build/run/error output loop. After a proposed change, the developer should be able to ask the AI Pad to interpret check/build output and suggest a correction.

```
Typical workflow:
1. Ask AI Pad for a change.
2. Review diff.
3. Approve apply.
4. Run Safire check/build.
5. Ask AI Pad to explain any error.
6. Approve a small fix.
7. Rebuild and record result.
```

Diagnostics

Diagnostics should be evidence-based. The AI Pad should read logs, source locations, support bundle files, license diagnostics, runtime output, and package manifests before giving a conclusion.

Support bundle workflow

20. Create a support bundle from SafireIDE.
21. Ask the AI Pad to summarize the bundle.
22. Identify build, runtime, licensing, deployment, or configuration issues.
23. Prepare a customer-facing support note if needed.
24. Prepare a developer action list for the next fix.

License diagnostics

The AI Pad should be able to help interpret license state messages in the IDE, while preserving the boundary that runtime and end-user packages should not depend on the licensed IDE.

12. Using the AI Pad with the Safire language

The AI Pad should understand Safire language objects as business artifacts. It should not treat Safire source as anonymous text. It should know common declaration types, naming conventions, binding rules, event styles, and source-first boundaries.

Core Safire source objects

Object	AI Pad should understand
Application	Title, version, startup window, default database, resources, culture, deployment metadata.
Window	Controls, layout, events, data binding, browse/update actions, validation, source write-back.
Control	Caption, binding, position/layout, masks, lookup, style, help, notes, events.
Table	Fields, keys, relationships, validation, default values, browse usage, report usage.
Query	Parameters, filters, joins, ordering, SQL-backed logic, report/browse consumers.
Report	Bands, fields, expressions, grouping, totals, parameters, page setup, preview/export.
Procedure	Business action, parameters, local data, return path, validation and persistence calls.
Class	Business service, private/public methods, reusable rules, persistence helpers.
Validation	Reusable business rules with user-facing messages.

Example: ask for a Safire window

```
Create a source-backed customer edit window.
Use Customer as the data source.
Include Code, Name, Email, CreditLimit, IsActive, Save and Cancel.
Keep event handlers short and move validation into a procedure.
Show the proposed Safire source only; do not apply it yet.
```

Example Safire source style

```
Window WIN_CustomerEdit
  Title = "Customer"
  Size = 520, 300
  DataSource = Customer

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

  Button BTN_Save
    Caption = "Save"
  End

  On BTN_Save.Click
    SaveCurrentCustomer()
  End
End
```

NOTE: The AI Pad should prefer readable Safire source, explicit names, short events, and business-oriented procedures.

13. AI-aware Safire source style

Why style matters

Good source style helps the compiler, IDE, designers, and AI assistant. Clear names and short handlers make project context easier to build and reduce the risk of misleading proposals.

Recommended conventions

Item	Recommended style
Windows	WIN_CustomerList, WIN_CustomerEdit, WIN_InvoiceBrowse.
Reports	RPT_Invoice, RPT_CustomerList, RPT_StockValuation.
Queries	Q_CustomerSearch, Q_OpenInvoices.
Buttons	BTN_Save, BTN_Close, BTN_Post.
Tables/Browses	TBL_Customers, BRW_Invoices.
Labels/Edits	LBL_Name, EDT_Name, EDT_CreditLimit.
Procedures	PostInvoice, SaveCustomer, RefreshCustomerBrowse.
Classes	CustomerService, InvoicePostingService.

Short event handlers

Events should connect UI action to business logic, not contain all business logic inline.

```
Good:
On BTN_Post.Click
   PostCurrentInvoice()
End

Avoid:
On BTN_Post.Click
   // Hundreds of lines of validation, posting, ledger, stock, and reporting logic.
End
```

Explicit bindings

Controls should declare bindings clearly so the IDE and AI Pad can understand how UI and data relate.

```
Edit EDT_CreditLimit
   Caption = "Credit limit"
   Binding = Customer.CreditLimit
   Format = "money"
End
```

14. Business UI and database workflows

Browse/update pattern

The AI Pad should understand browse/update as a first-class business application pattern: browse a list, select a row, open an update window, validate, save, refresh the browse.

```
Browse window -> Query -> Table rows -> New/Edit/Delete/View actions -> Update window -> Validation -> Save -> Refresh browse
```

Prompt recipe: add a field to a business screen

```
Add Customer.PhoneNumber to the customer maintenance flow.
Find the table, browse, edit window, validation, and related reports.
Prepare a patch proposal but do not apply it until I review the diff.
```

AI Pad should check

- Dictionary/table field declaration.
- Browse columns and search behavior.
- Edit window control and binding.
- Validation rule if required.
- Reports or exports that should show the new field.
- Database migration or data update implications.
- Build/check result after approved apply.

Prompt recipe: improve validation

```
Review customer save validation.
Move reusable business rules into a Validation block or service procedure.
Keep the Save button event short.
Show me the proposed diff before applying.
```

15. Reports, deployment, and maintenance workflows

Report assistance

Reports are business artifacts. The AI Pad should understand report source, bands, fields, expressions, grouping, totals, page setup, preview/export paths, and related queries.

```
Explain RPT_CustomerList.
List its data source, parameters, bands, fields, grouping, totals, and export behavior.
Tell me which query and table fields it depends on.
```

Report change prompt

```
Add a Region subtotal to RPT_CustomerList.
Check the query, report grouping, totals, and page footer.
Prepare a small patch and show the diff before applying.
```

Deployment assistance

The AI Pad should help developers understand deployment packages, runtime dependencies, support bundles, and product health checks. End-user deployments should not require SafireIDE.

```
Review the latest end-user deployment package evidence.
Confirm that the package contains the application executable, runtime files, config, report/export support, and no licensed IDE payload.
```

Maintenance workflow

The AI Pad should support long-term maintenance by explaining old project areas, documenting decisions, producing change summaries, and helping create release notes from applied audit entries.

16. Prompt recipes for daily use

The following prompts are intended for everyday SafireIDE AI Pad use. They assume the pad has access to the active project through Safire-controlled context building.

Goal	Prompt
Understand a window	Explain WIN_CustomerList. Include controls, bindings, events, source file, query, and related procedures.
Find business logic	Where is invoice posting implemented? Search procedures, classes, events, validation, and reports.
Prepare a safe change	Add a PhoneNumber field to Customer. Prepare a patch proposal only. Show affected files and diff.
Diagnose build failure	Read the latest build output. Explain the first real error, likely cause, and smallest safe fix.
Improve event style	Find long event handlers in this window and propose moving business logic into procedures.
Review a report	Summarize RPT_Invoice, including data source, bands, totals, parameters, and export path.
Prepare release notes	Use review history and build results to draft technical release notes for this build.
Check licensing boundary	Review package evidence and confirm no licensed IDE payload is included in the end-user deployment.
Create documentation	Create a user-facing description of this customer maintenance screen from the source.
Analyze support bundle	Summarize this support bundle and identify likely configuration, runtime, or license issues.

Good prompt structure

1. Name the artifact or business goal.
2. Tell the AI Pad whether to explain, propose, or apply.
3. Ask for source files and evidence used.
4. Require a diff before changes.
5. Ask for build/check after approved apply.

17. Security, governance, and operational controls

Project control

The AI Pad must operate through Safire-controlled project services. It should not have uncontrolled write access, uncontrolled shell access, or the ability to silently edit source.

Human approval

A mature business system should not be changed without developer review. The AI Pad can suggest and prepare, but approval remains with the developer.

Secrets and customer data

- Do not put API keys, passwords, or database passwords directly in prompts or source.
- Use protected configuration or environment variables for model endpoint authentication.
- Use local model options for sensitive customer environments where policy requires it.
- Redact customer data from support bundles when required.
- Keep audit logs useful but avoid storing unnecessary confidential data.

Audit and accountability

Every AI-assisted source change should have a clear record: what was requested, what files were changed, who approved it, where backups are stored, and whether the build/check passed afterward.

Team policy suggestions

Policy area	Suggested rule
Read-only exploration	Allowed for all developers with project access.
Patch proposals	Allowed for developers who can edit source.
Approved apply	Allowed only for developers with write permission.
Build/test execution	Allowed for developers with build permission.
Deployment packaging	Restricted to release owners.
License diagnostics	Allowed for support and license administrators.

18. Adoption plan for a business development team

Phase 1: Read-only assistance

- Use the AI Pad to explain project structure.
- Ask it to summarize windows, reports, tables, and build output.
- Use it to create developer notes and onboarding summaries.
- No AI-applied source changes yet.

Phase 2: Proposal-only changes

- Allow patch proposals but require manual review.
- Use diff preview for every change.
- Keep proposals small and focused.
- Use build/check after manual application or approved apply.

Phase 3: Approved apply with backups

- Enable approved apply for trusted developers.
- Require backup and audit evidence.
- Require build/check proof before closing work.
- Use review history for rollback and release notes.

Phase 4: Support and release workflow

- Use support bundles as standard customer evidence.
- Use AI Pad to summarize diagnostics and logs.
- Use audit trail to prepare release notes.
- Use package checks to confirm licensed IDE boundary and runtime completeness.

19. Troubleshooting and safe operating practices

Common problems

Problem	Safe response
AI Pad answers generically	Build project context first and ask a more specific artifact-based question.
Search results show generated files first	Restrict search to user-owned source or ask for generated evidence separately.
Patch touches too many files	Reject the proposal and ask for a smaller patch.
Diff modifies generated cache	Reject unless the generated file is intentionally owned as source.
Build fails after apply	Read first real compiler error, restore if necessary, then propose a minimal fix.
Model endpoint unavailable	Check endpoint URL, token, timeout, local service status, and SafireAgentHost diagnostics.
License state blocks IDE command	Open license diagnostics and support bundle; confirm runtime/CLI boundary.

Safe operating rules

- Use explanation mode before applying changes in unfamiliar project areas.
- Prefer small patches that solve one business problem at a time.
- Keep event handlers short and business logic centralized.
- Run check/build after approved source edits.
- Review audit history before packaging a release.
- Restore from backup rather than hand-reversing a complex unwanted patch.

When not to use AI apply

CAUTION: Avoid AI apply for legal/compliance calculations, payroll/tax rules, database migrations, or customer-critical posting logic unless the developer can fully review and test the result.

20. Complete example: assisted customer browse/update change

Business request

A customer asks for the customer maintenance screen to capture a phone number, show it in the browse, and include it in the customer list report.

Developer prompt

```
Add Customer.PhoneNumber to the customer maintenance flow.
Find the table, query, browse window, edit window, validation, and customer list report.
Prepare a patch proposal only. Show affected files and diff before applying.
```

Expected AI Pad context

- dictionary/Customer.sfdict or equivalent table source.
- queries/Q_CustomerSearch.sfquery.
- windows/WIN_CustomerList.sfwin.
- windows/WIN_CustomerEdit.sfwin.
- reports/RPT_CustomerList.sfrpt.
- Any related validation or service procedure source.

Expected proposal summary

- Add Field PhoneNumber is String(30) to Customer.
- Include PhoneNumber column in customer browse where useful.
- Add EDT_PhoneNumber with binding to Customer.PhoneNumber.
- Add optional validation or formatting rule if required by project convention.
- Add report field or adjust layout in customer list report.
- Run check/build after approved apply.

Approval checklist

- Diff touches only expected source files.
- No generated cache is edited as primary source.
- Field name and UI caption match project naming style.
- Report layout still fits page width.
- Build/check passes or produces clear next action.

21. Complete example: assisted report enhancement

Business request

Management wants the monthly sales report to group by region and show region totals.

Developer prompt

```
Review RPT_MonthlySales and its data source.
Add grouping by Region and region totals.
Check whether the query includes Region.
Prepare a small patch proposal and show the diff before applying.
```

Expected AI Pad analysis

- Identify the report source file.
- Identify the query or SQL used by the report.
- Check whether Region is available from the data source.
- If missing, propose query adjustment before report band changes.
- Add group header and footer with total expression.
- Check page layout and formatting.

Example report source fragment

```
Group By Sales.Region
  Header
    Text Sales.Region At 10mm, 0mm Bold
  End
  Footer
    Text "Region total:" At 120mm, 0mm
    Total Sales.Amount Sum At 160mm, 0mm Format "money"
  End
End
```

Post-change proof

After approved apply, the developer should preview/export the report, run the relevant build/check step, and store the result in review history or release evidence.

22. Appendix A: quick workflow reference

Workflow	Steps
Ask about project	Open project -> Open AI Pad -> Build context -> Ask artifact-specific question -> Review answer.
Propose change	Ask for proposal only -> Review affected files -> Review diff -> Approve or reject.
Apply change	Approve apply -> Safire creates backup -> Audit entry created -> Run check/build.
Restore change	Open review history -> Select applied entry -> Preview restore diff -> Approve restore -> Run check/build.
Diagnose build	Open build output -> Ask AI Pad to explain first real error -> Propose minimal fix.
Prepare release notes	Review applied entries -> Summarize changes -> Include build/package proof.
Support issue	Create support bundle -> Ask AI Pad to summarize -> Identify action list.

Decision rule

NOTE: Explain first. Propose second. Diff before apply. Backup before write. Build after change. Audit always.

23. Appendix B: glossary

Term	Meaning
AI Assistant Pad	Dockable SafireIDE panel for project-aware AI assistance.
SafireAgentHost	Safire-controlled service that handles source search, context, patch proposal, approved apply, backup, and build/test actions.
Source-first	Rule that developer-owned source files remain the authoritative application truth.
Patch proposal	A proposed set of source changes shown for review before application.
Diff	Visual comparison of existing source and proposed source.
Approved apply	Developer-approved source update performed through Safire-controlled tooling.
Backup	Saved copy of changed files before apply or restore.
Audit trail	Record of AI request, proposal, approval, changed files, backup, and build/test result.
Review history	SafireIDE panel or record showing past source write-back and restore actions.
Runtime plan	Generated runtime representation used for preview/runtime behavior; not the primary source truth.
Licensed IDE	Commercial SafireIDE productivity product.
Runtime package	Files required for an end user to run a deployed Safire application.

24. Appendix C: AI Pad readiness checklist

- Active project root is detected correctly.
- User-owned source folders are indexed.
- Generated/build folders are marked as generated evidence.
- Model endpoint is configured and authenticated where required.
- AI session audit path is writable.
- Patch proposal folder is writable.
- Backup folder is writable.
- Diff preview opens before apply.
- Approved apply requires explicit developer confirmation.
- Build/check command can be run from SafireIDE.
- Review history can show applied entries.
- Restore preview and approved restore path are available.
- License diagnostics can be opened when needed.
- Runtime/CLI independence from licensed IDE gating is preserved.

25. Closing note

The SafireIDE AI Assistant Pad should become one of Safire's strongest commercial differentiators because it is tied to the business application model rather than acting as a detached chatbot. Its value comes from knowing what a Safire project is, respecting source ownership, and helping experienced developers move faster without surrendering control.

For the Safire language, the AI Pad should encourage readable source, short event handlers, explicit data binding, source-backed windows and reports, clear database and dictionary declarations, validation rules, buildable projects, and deployable packages. For SafireIDE, it should provide calm, traceable assistance that fits the way mature business developers already think.