aws

Developer Guide

AWS Coding Plugin for SAP ABAP



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Coding Plugin for SAP ABAP: Developer Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is it? 1
Key features 1
How AWS Coding Plugin for SAP ABAP works 2
Basic concepts 2
Use cases 3
Pricing 3
Compliance and security 3
Related services 3
Next steps 4
How it works 5
Architecture 5
How AWS Coding Plugin for SAP ABAP works with Eclipse 6
How AWS Coding Plugin for SAP ABAP works with Kiro 6
How Kiro reaches your SAP system 7
Working with ABAP 7
What the agent can do in your ABAP system 8
Areas the agent works in 8
Eclipse integration 9
Limitations 9
ABAP development 9
Creating 10
Editing 10
The self-correcting loop 10
Kiro Permissions 10
Getting started 11
Prerequisites 11
Install the plug-in and run your first spec 11
Troubleshooting installation issues 13
Using the plug-in 14
Using chat 14
Opening the chat panel 14
Sending a prompt 14
Working across multiple tabs 15
The selector bar 15

Monitoring credit usage ... 16
Copying content from a conversation ... 16
Managing context ... 16
Editor actions ... 17
Context menu actions ... 17
Inline assist ... 17
Spec-driven development ... 18
Why spec-driven development matters for SAP ABAP ... 18
The three spec documents ... 18
Working with specs in Eclipse ... 19
How specs are written ... 20
Change management ... 20
Working with transports ... 20
Reviewing and reverting changes ... 20
Best practices ... 21
Customizing ... 23
Steering ... 23
Built-in ABAP steering ... 23
Organization steering ... 23
Per-system steering ... 23
Writing your own steering ... 23
Preferences reference ... 24
AWS Coding Plugin for SAP ABAP preferences ... 24
Theme, fonts, and colors ... 26
Updates and maintenance ... 27
Check for plug-in updates ... 27
Review release notes ... 27
Uninstall AWS Coding Plugin for SAP ABAP ... 27
Security ... 29
The plug-in and Eclipse ... 29
The plug-in and Kiro ... 29
What data leaves your machine ... 29
Kiro and the MCP servers ... 30
Reaching your SAP system ... 30
Credential and token handling ... 31
Troubleshooting ... 32

Common issues 32
Additional help 33
Logging 33
What is recorded at INFO level 33
What is recorded at DEBUG level 34
Additional logging settings 34
Collecting diagnostics for a support request 34
Pricing 36
FAQ 37
Getting started 37
Working with SAP 37
Security and data 39
Document history 41

What is AWS Coding Plugin for SAP ABAP?

AWS Coding Plugin for SAP ABAP ("Coding Plugin") is an Eclipse plug-in that adds agentic AI development, powered by Kiro, to the SAP ABAP Development Tools (ADT) environment you already use. It provides an AI chat panel, spec-driven development workflows, editor actions, and direct access to your SAP system through your existing ADT connection. For Kiro's own behavior, including its security model, sign-in, and steering semantics, see the Kiro documentation on the Kiro website.

There is no charge for the plug-in. Using it consumes Kiro credits from your Kiro plan. See *Pricing*. The plug-in reaches your SAP system through your own ADT connection, using your SAP user and authorizations. See *Security*.

Key features

Coding Plugin provides the following:

- **Agentic chat in Eclipse.** A chat panel with multiple tabs where you prompt Kiro and review the resulting changes without leaving the IDE.
- **Spec-driven development for SAP ABAP.** Kiro generates requirements, design, and task documents, then works through the tasks.
- **Direct access to your SAP system.** Through your existing ADT connection, Coding Plugin finds, reads, changes, and activates ABAP objects, runs ABAP Unit tests and ABAP Test Cockpit (ATC) checks, and works with transports.
- **Editor integration.** Context menu actions (Explain, Fix, Optimize, Refactor), inline assist, and clickable `adt://` links that open an object in the Eclipse editor.
- **An Eclipse-aware context.** Coding Plugin knows which object you have open, reads the Problems view, and opens editors and views on request.
- **Kiro Permissions in Eclipse.** You approve each action Kiro takes, surfaced as prompts in the Eclipse UI.
- **Built-in ABAP steering.** The plug-in ships ABAP-aware steering so Kiro's output fits the way SAP ABAP is developed.

How AWS Coding Plugin for SAP ABAP works

Coding Plugin runs inside Eclipse, alongside SAP ADT. When you send a prompt from the chat panel, the plug-in passes it to Kiro CLI on your machine, which calls Kiro's cloud services for model inference. The plug-in gives Kiro a set of MCP tools that reach your SAP system through the ADT connection you have already defined, so it can find objects, read and change source, activate, run ABAP Unit tests and ATC checks, and record changes on a transport. These tools are governed by Kiro's permission model to ask for approvals from the developer.

For more information about the components involved and how they communicate, see How AWS Coding Plugin for SAP ABAP works and Architecture.

Basic concepts

The following terms describe how you work with Coding Plugin:

Eclipse and SAP ADT

Eclipse is an open source integrated development environment (IDE). SAP ABAP Development Tools (ADT) is a set of Eclipse plug-ins that let you develop ABAP in Eclipse instead of SAP GUI. For more information, see ABAP Development Tools for Eclipse in the SAP documentation.

Agent

Kiro is the agent. It reasons about your prompt and acts by calling tools, such as reading an ABAP object or running a test.

Spec

A spec is a set of requirements, design, and task documents that Kiro writes and then works through, so that you plan a large change before making it. See the section called "Spec-driven development".

Model Context Protocol (MCP) server

An MCP server exposes a set of tools to Kiro. The plug-in wires the SAP-specific MCP servers automatically, so there is nothing to configure. See Model Context Protocol (MCP) in the Kiro documentation.

Steering

Steering is written guidance that shapes how Kiro works. The plug-in ships ABAP-aware steering out of the box, and you can add your own. See *Customizing*.

Kiro Permissions

Kiro Permissions is Kiro's approval mechanism. By default, every tool call waits for your approval. See the section called "Kiro Permissions".

Use cases

Coding Plugin is designed to be a general-purpose tool that can satisfy a broad set of use cases for SAP ABAP, rather than being targeted at a specific use case. The following are examples of things Coding Plugin can do:

- Build a new RICEF object (report, interface, conversion, enhancement, or form), Open Data Protocol (OData) service, or other feature from a spec, with Coding Plugin discovering the available tables and APIs in your live system.
- Add unit tests to existing code and then refactor it for clean core or S/4HANA, iterating through ATC checks.
- Understand or document legacy code.
- Perform code reviews for style or correctness.

Pricing

There is no separate charge for Coding Plugin. The Kiro work you do through it draws credits from your Kiro plan. For more information, see AWS Coding Plugin for SAP ABAP pricing.

Compliance and security

Coding Plugin reaches your SAP system only through the ADT connection you have already defined in Eclipse, using your SAP user and your SAP authorizations. It stores no SAP credentials of its own, and it can do nothing on the system that your SAP user cannot do. Tool calls are subject to Kiro's permission model. For more information about how the plug-in's MCP server is secured, and about Kiro's own security model, see Security.

Related services

- Kiro is the agent that powers Coding Plugin. For Kiro's own behavior, models, and plans, see the Kiro documentation on the Kiro website.

- With the AWS SDK for SAP ABAP, ABAP programs can call AWS services directly from ABAP. See What is AWS SDK for SAP ABAP?.

Next steps

- For instructions on installing the plug-in and running your first spec, see Set up AWS Coding Plugin for SAP ABAP.
- For more information about how specs work in ABAP, see Spec-driven development.
- For more information about what Coding Plugin can do in your SAP system, see Working with ABAP.

How AWS Coding Plugin for SAP ABAP works

AWS Coding Plugin for SAP ABAP ("Coding Plugin") brings agentic development, powered by Kiro, to the Eclipse environment and SAP ABAP Development Tools (ADT) that you already use. It runs inside Eclipse as a plug-in. It adds an agentic chat window with Kiro-powered spec-driven development workflows, launches and manages Kiro CLI, connects it to a set of built-in MCP servers, and reaches your SAP system through the SAP ADT SDK over your existing ADT connection. By integrating with both Eclipse and Kiro, it makes Kiro a true partner of the human developer throughout the SAP ABAP development process.

Topics

- Architecture
- Working with ABAP
- ABAP development
- Kiro Permissions

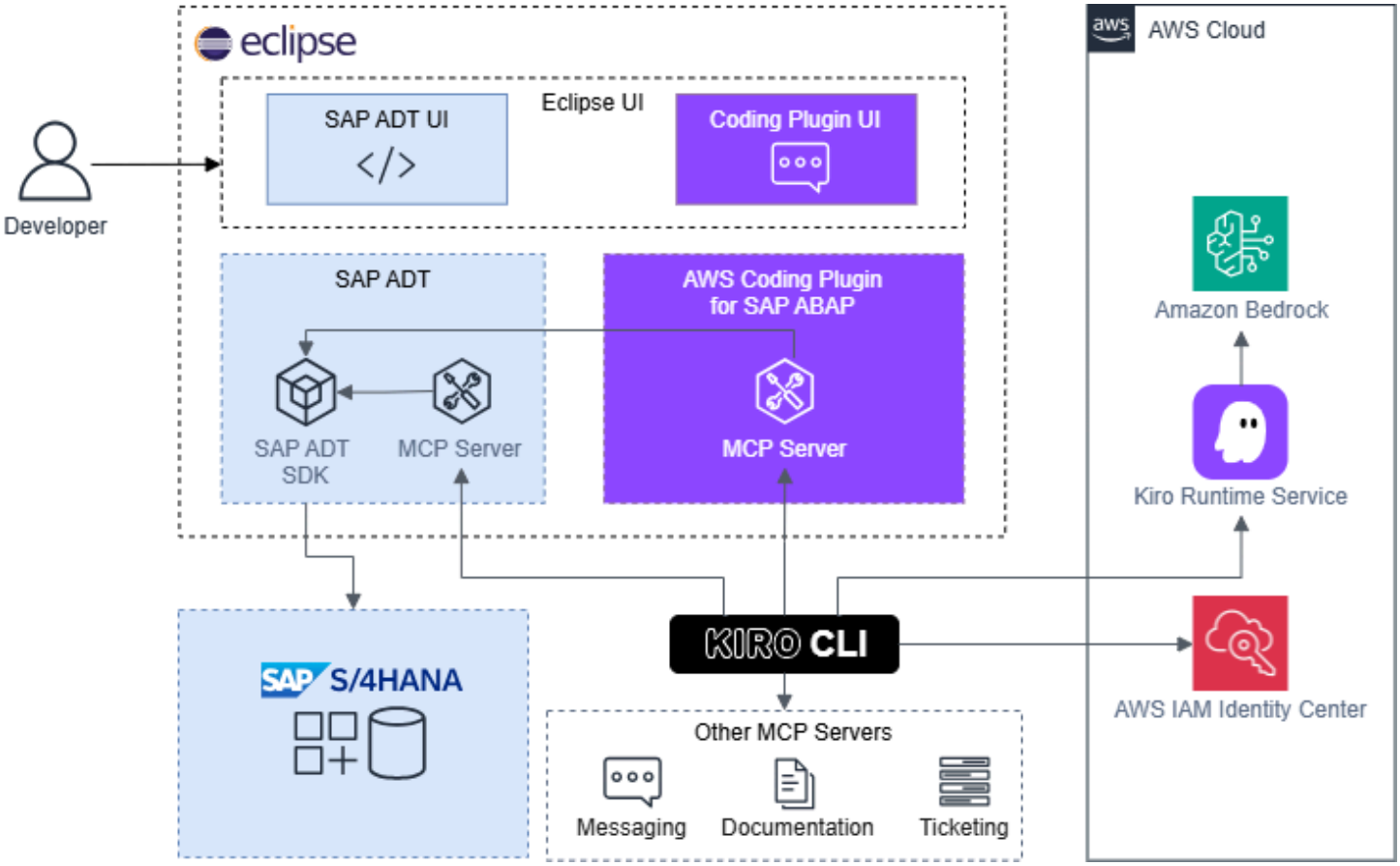
Architecture

The architecture consists of five connections:

- AWS Coding Plugin for SAP ABAP to Kiro CLI, using an stdio pipe and the Agent Client Protocol (ACP) to exchange prompts, statuses, and permission requests
- AWS Coding Plugin for SAP ABAP to SAP ADT via the SAP ADT SDK
- Kiro CLI to the Kiro backend service and to your identity provider
- Kiro CLI to the MCP servers, using the MCP protocol
- The MCP servers to the SAP system, using the SAP ABAP Development Tools (ADT) SDK

The following diagram shows these connections. The Kiro backend is the Kiro Runtime Service, which runs models on Amazon Bedrock. The identity provider shown, AWS IAM Identity Center, is one of several sign-in options that Kiro supports. The dashed box stands for other MCP servers you can add, such as messaging, documentation, or ticketing servers.

Coding Plugin architecture



SAP S/4HANA is the trademark or registered trademark of SAP SE or its affiliates in Germany and in other countries.

Eclipse is a trademark of Eclipse Foundation AISBL.

How AWS Coding Plugin for SAP ABAP works with Eclipse

Coding Plugin is an Eclipse plug-in, so it runs inside your Eclipse environment and has access to Eclipse's APIs. The Coding Plugin MCP server gives Kiro the same Eclipse access, creating a two-way connection to the IDE. Kiro can read Eclipse's Problems view, see which object you have open in the editor, and open editors and additional Eclipse views on request. Object references in the chat become clickable `adt://` hyperlinks that open the object in its ADT editor.

How AWS Coding Plugin for SAP ABAP works with Kiro

Coding Plugin automatically launches Kiro CLI in the background and connects it to the two built-in MCP servers. All agentic processing happens in Kiro CLI, which supports the full range of Kiro capabilities, such as steering, skills, and specs. The integration is bidirectional: the plug-in sends

your prompts and the surrounding context to Kiro, and surfaces Kiro's responses, permission prompts, and actions in the Eclipse UI.

MCP servers. AWS Coding Plugin for SAP ABAP automatically connects the SAP ADT MCP server and the AWS Coding Plugin for SAP ABAP MCP server, and they do not need to be included in `mcp.json`. Through these two servers, Kiro can do the same things you can do in Eclipse and SAP ADT, using the same mechanisms. You can configure your own additional MCP servers through `mcp.json` files. See Model Context Protocol (MCP) in the Kiro documentation.

- *SAP's ADT server* is SAP's first-party MCP server. Coding Plugin is designed to favor SAP's ADT server as the canonical and preferred source of ADT functionality.
- *The plug-in's own server* is vastly more functional than SAP's server and uniquely optimized for agentic use. It includes a broad set of ADT tools, Eclipse integration (reading the Problems view, seeing which object you have open, opening editors), and additional functions for troubleshooting the plug-in itself. The two servers overlap somewhat; for example, both can activate objects, run ABAP Unit tests and ABAP Test Cockpit (ATC) checks, and manage transports.

How Kiro reaches your SAP system

Kiro CLI and AWS Coding Plugin for SAP ABAP do not establish a separate connection to your SAP system. They reach it through the SAP connection you already have defined in Eclipse in SAP ADT. Your existing network path, authentication, and SAP authorizations apply. Kiro CLI sends your prompts and the context it gathers to the Kiro backend service, which runs the model. For more information on Kiro data handling, see Privacy and security in the Kiro documentation. AWS Coding Plugin for SAP ABAP has no service endpoint of its own.

Working with ABAP

AWS Coding Plugin for SAP ABAP ("Coding Plugin") works directly in your SAP ABAP system through the SAP ABAP Development Tools (ADT) connection you already use in Eclipse. These ABAP tools come from SAP's first-party ADT Model Context Protocol (MCP) server, the canonical and preferred source of ADT functionality, and from the plug-in's own MCP server, which is vastly more functional and uniquely optimized for agentic use. The two overlap somewhat (see Architecture).

What the agent can do in your ABAP system

Coding Plugin works with SAP ABAP objects the way you do in SAP ADT: it finds objects, reads their source, changes and activates them, verifies them, and places them on transports. It uses your existing ADT connection, so every action runs as your SAP user. Coding Plugin also verifies its own work through the self-correcting loop. It activates the object and runs ABAP Unit tests and ABAP Test Cockpit (ATC) checks, then corrects what it finds and repeats (see The self-correcting loop).

Every tool call is gated by Kiro Permissions, Kiro's per-call approval mechanism (see Kiro Permissions), and is bounded by your SAP user's own authorizations.

Areas the agent works in

Coding Plugin works across these areas of your ABAP system. For how a whole object is read, why large objects go through a local file, and how a change stays inactive until you activate it, see ABAP development.

- **Find and navigate.** Search the repository by name pattern, object type, or package, find where an object is used, and jump to a definition or its references.
- **Source code.** Read and edit the source of classes, programs, function modules, Core Data Services (CDS) views, and other object types. Run a syntax check. Format with the ABAP pretty printer or the CDS formatter. Lock and unlock objects.
- **Object creation.** Create classes, interfaces, programs, function groups and modules, DDIC objects (domains, data elements, structures, tables, and table types), CDS views, ABAP RESTful Application Programming Model (RAP) artifacts, packages, and transformations.
- **Activation.** Activate objects and list what is still inactive.
- **Transports.** Check whether an object needs a transport, search transports, create one and add objects, and set the owner. For releasing, reviewing, and reverting work, see Managing change and code reviews.
- **Version control.** Read native ABAP version history, and work with abapGit. abapGit support is available by default in BTP ABAP.
- **Runtime errors.** List and read ABAP short dumps, including the program, timestamp, category, runtime error, and call stack.
- **Data.** Run read-only SQL queries and preview the contents of tables and CDS views.
- **RAP and OData services.** Generate RAP objects, and define, bind, and publish Open Data Protocol (OData) services.

- **System information.** Report installed components, system and client details, and the system's users.
- **SAP GUI.** Display, edit, or run an object, or start a transaction. This requires SAP GUI to be installed.

Eclipse integration

Coding Plugin makes Kiro aware of your Eclipse workbench. Through the plug-in's own MCP server (see Architecture), Kiro has a two-way connection to the IDE. It reads the state of your workbench and acts on it with your approval through Kiro Permissions (see Kiro Permissions).

- **Active editor.** Coding Plugin reports which object is open. The active editor is pinned to your prompt, so a request such as "Run an ATC check on this program" understands what you are referring to.
- **Opening objects and views.** Coding Plugin opens an ABAP object in its ADT editor, and opens other Eclipse views, on request. This helps it walk you through a code review.
- **Project Explorer.** The Coding Plugin creates a filesystem-based project in the Project Explorer. This project contains the `.kiro/` folder and makes it visible, so its settings, steering, skills, and specs sit alongside your ABAP projects.
- **Problems view.** Coding Plugin reads the syntax errors and warnings Eclipse reports on an object.

Limitations

The ADT services available depend on the connected SAP system.

- Older systems expose fewer ADT services, so certain functionality is not available. For example, the system refuses transport release on systems whose ADT cannot release transports (such as SAP NetWeaver 7.40). Coding Plugin offers SAP's own release dialog instead.
- The SAP BTP ABAP environment runs ABAP Cloud and supports a different set of functionality than a standard on-premises SAP system.

ABAP development

Developing ABAP objects differs from classic Git-based development workflows. ABAP objects are not files, and a change becomes an inactive version in the live system.

Creating

Manipulating objects through ABAP Development Tools (ADT) does not follow filesystem semantics. In particular, creating an object always requires additional metadata such as specifying a package and a transport. ADT does not support partial source writes, so updating an object replaces the entire object.

Editing

To change an object, Coding Plugin retrieves the entire object, because ADT has no partial read. For large objects, Coding Plugin writes that source to a temporary local file so that Kiro can search, edit, and append with its built-in file commands, instead of loading the whole object into the model's context. This ensures high fidelity when editing objects and conserves the session's context window. The edited object becomes an inactive version in the SAP system until it is activated as a separate operation.

The self-correcting loop

AWS Coding Plugin for SAP ABAP gives Kiro knowledge of the *self-correcting loop* used in ABAP development: activation, unit tests, and ABAP Test Cockpit (ATC) checks that verify syntax, preserved functionality, and ABAP code quality. This loop lets Coding Plugin find and correct its own errors, and is foundational to agentically generating high-quality code.

Kiro Permissions

Kiro Permissions is Kiro's core safety mechanism, and AWS Coding Plugin for SAP ABAP surfaces it in the Eclipse UI for the developer. By default, every tool call is gated through an explicit approval prompt. An approval covers exactly one tool call unless the approver chooses to apply a pattern to the approved resource. The permission model itself (the choices, the scopes, and how rules are matched) is built into Kiro. See Permissions in the Kiro documentation.

Getting started with AWS Coding Plugin for SAP ABAP

This tutorial takes you from an empty Eclipse workspace to your first spec running against your SAP system. The setup takes a few minutes, not counting Kiro CLI installation and sign-in if you have not done those already.

Completing this setup requires a Kiro plan. There is no charge for the plug-in; running it consumes Kiro credits from your plan. See *Pricing*.

Topics

- Prerequisites
- Install the plug-in and run your first spec
- Troubleshooting installation issues

Prerequisites

Before you start, make sure you have:

- An SAP ABAP system. Coding Plugin is tested with SAP ERP 6.0 EHP7, SAP S/4HANA 2025, and the SAP BTP ABAP environment. Available ABAP Development Tools (ADT) functions vary by `SAP_BASIS` release. Older systems support reading and writing code but might lack some advanced functions. See Limitations.
- Eclipse 2026-06 or later, with SAP ADT 3.60.0 or later.
- An SAP connection already defined and open in ADT.
- Kiro CLI 2.19.0 or later, installed and signed in to Kiro.
- A Kiro plan.

Install the plug-in and run your first spec

1. **Install the plug-in.**
   a. In Eclipse, choose **Help**, **Install New Software**.
   b. In **Work with**, enter `https://downloads.aws.com/sap/aws-coding-plugin-for-sap-abap/current`.
   c. Select the AWS Coding Plugin for SAP ABAP feature, then choose **Next**.

d. Accept the license terms, then choose **Finish**.

e. Restart Eclipse when prompted.

The plug-in loads and the onboarding wizard opens.

2. **Complete the onboarding wizard.** The wizard checks your prerequisites and locates Kiro CLI.

a. If the wizard reports that Kiro CLI is not found, install it from Kiro CLI on the Kiro website, then choose **Check Again**.

b. When the wizard starts Kiro sign-in, sign in with your Kiro account.

The wizard reports that all prerequisites are satisfied.

3. **Open the chat panel.**

a. Press **Ctrl+3** (macOS: **Cmd+3**) and open the AWS Coding Plugin Chat view.

4. **Automatic project configuration.**

a. The first time you open the plug-in, it creates a project that holds a `.kiro/` folder (with `settings/`, `skills/`, `specs/`, and `steering/`). This project is the agent's working directory and can be committed to Git.

b. The plug-in also runs two setup commands automatically:

- `/workspace-init` interviews you about your organization's standards and writes them as steering.
- `/sap-discover` interviews you about the connected SAP system and writes system-specific steering.

Both can be run again later to update what they wrote. For more information, see Steering.

5. **Run your first spec.**

a. In the chat, describe the change you want to make in your SAP ABAP system.

b. When Kiro proposes a spec, review the generated requirements, design, and tasks.

c. At the top of the chat tab, choose **Run All Tasks**.

d. Approve each tool call when its prompt appears.

Coding Plugin opens a new tab and reports progress one task at a time. Your spec is saved under `.kiro/specs/` in the project.

By default, Kiro gates each tool call behind an approval prompt that you accept or decline (Kiro Permissions). For how approvals work, see Kiro Permissions.

Troubleshooting installation issues

- **The wizard cannot find Kiro CLI.** Install Kiro CLI, then choose **Check Again** in the wizard.
- **The SAP features do not appear.** Your SAP ADT is older than 3.60.0. Update ADT to 3.60.0 or later.

Using AWS Coding Plugin for SAP ABAP

After you set up Coding Plugin, your day-to-day work happens in the chat panel and in the Eclipse editor. This chapter covers the workflows you use to get things done.

Topics

- Using the AWS Coding Plugin for SAP ABAP chat panel
- Editor actions
- Spec-driven development with AWS Coding Plugin for SAP ABAP
- Managing change and code reviews
- Best practices

Using the AWS Coding Plugin for SAP ABAP chat panel

The Coding Plugin chat panel is where you work with Kiro inside Eclipse. You open it as an Eclipse view, type a prompt, and Kiro replies in the same panel. With separate tabs, you can hold more than one conversation at a time. With the selector bar, you can set the model and mode Kiro uses.

Opening the chat panel

The chat panel opens as an Eclipse view, and holds each conversation in a tab in that view.

To open the chat panel

1. Choose **Window**, **Show View**, **Other...**.
2. Choose AWS Coding Plugin Chat, and then choose **Open**.

You can also open the panel from Quick Access: press **Ctrl+3** (macOS: **Cmd+3**), type AWS Coding Plugin Chat, and then press **Enter**.

Sending a prompt

With the chat panel open, you send a prompt from the input field at the bottom of the panel.

To send a prompt

1. Enter your prompt in the input field at the bottom of the panel.

2. (Optional) Choose up to five documents to attach to the prompt, so Kiro has more context. To work around this limit, Kiro can also read files directly from your filesystem.
3. Choose **Send**, or press **Enter**.

To write a prompt that spans several lines, press **Shift**+**Enter** to start a new line without sending the prompt.

Kiro responds in the same tab. The response streams in as Kiro works, and any permission prompt for a tool call appears inline in the conversation (see Kiro Permissions).

Working across multiple tabs

Each tab holds its own conversation with Kiro. You can open up to 10 tabs and run them at the same time. While Kiro works in one tab, you can send a prompt in another. A reply always stays on the tab you sent it from. Closing a tab stops the work running in it. When you reopen the chat panel, it restores your most recently used tabs.

There are three exceptions to the parallelism between tabs:

- **A pending permission prompt.** When a tab is waiting for you to approve or deny a tool call, you cannot submit a new prompt in another tab. A banner directs you to respond to the pending prompt first.
- **A running spec.** Kiro runs one spec at a time, so while a spec task is executing it serializes the work in your other tabs.
- **A turn already running in another tab.** Kiro runs one turn at a time across all tabs, so a prompt you submit while another tab's turn is running waits for that turn to finish. Until yours starts, the Coding Plugin shows `Waiting — another tab's turn is running…`.

The selector bar

A selector bar at the bottom of the chat panel lets you choose the model and mode Kiro uses for a conversation, and shows how much the conversation costs to run.

The model and mode lists are a view onto what your Kiro backend offers, so the available choices come from Kiro, not from the plug-in. For what the models are and how they differ, see Models in the Kiro documentation.

Kiro's models are general-purpose and are not tuned specifically for SAP ABAP, but the larger, more capable models show strong knowledge of SAP ABAP coding practices. SAP ABAP development involves many tool calls across many objects, so prefer a model with a context window of about 1 million tokens. A stronger model can do better on reasoning-heavy work such as authoring a spec. A smaller, lower-cost model is often enough for implementing the spec. This is a fast-moving area, so experiment to find the models that work best for you.

Monitoring credit usage

Coding Plugin shows how your Kiro credits are being spent in several places:

- **Total.** A credit meter at the bottom of the window shows your account's overall usage.
- **Per tab.** Hovering over a tab header shows that conversation's credit usage in a tooltip.
- **Per turn.** Each turn's credit usage is shown on the tab.
- **Per spec.** Coding Plugin tracks estimated credit usage for each spec. This is not shown in the UI. Ask Kiro about a spec's usage to see it.

For what a credit is and how usage maps to your Kiro plan, see Pricing.

Copying content from a conversation

You can copy content out of a conversation at three levels:

- **A code block.** Each code block in a reply has a **Copy** button that copies that block's code to the clipboard.
- **A turn.** Each turn has a **Copy as Markdown** button in its footer that copies that one exchange as Markdown.
- **The whole conversation.** Open the context menu for the tab's heading and choose **Copy entire transcript** to copy the full conversation to the clipboard.

Managing context

A context meter in the chat panel shows how much of the conversation's context window is in use. As a conversation grows, Kiro compacts its earlier history to stay within that window, so a long conversation can continue without you starting over. For more on how Kiro manages context and compacts history, see Compaction in the Kiro documentation.

Editor actions

AWS Coding Plugin for SAP ABAP ("Coding Plugin") brings Kiro into the editors in Eclipse. You act on code without leaving the editor: run an action on a selection from the context menu, or modify a selection with inline assist.

Context menu actions

Select code in an editor, open the context menu, and choose the AWS Coding Plugin submenu. The submenu offers five actions:

- **Explain** asks Kiro to describe the selected code. The response appears in the chat panel.
- **Fix**, **Optimize**, and **Refactor** each ask Kiro to rewrite the selection. The proposed change appears as an inline diff in the editor.
- **Send to Prompt** copies the selection into the chat input so you can add an instruction before submitting.

To use a context menu action:

1. Open an ABAP object or a text file in the editor.
2. Select the code you want to act on.
3. Open the context menu for the selection, choose AWS Coding Plugin, and then choose the action.

Inline assist

Inline assist changes a selection in place, without going to the chat panel.

1. Select the code you want to change.
2. Press **Ctrl**+**Shift**+**I** (macOS: **Cmd**+**Shift**+**I**) to open the inline assist input next to the selection.
3. Type an instruction and press **Enter**.
4. Review the inline diff, then choose **Accept** or **Reject**.

Accept replaces the selection with the proposed change. **Reject** or **Esc** restores the original code. If Coding Plugin cannot parse the response into code, it declines the change and restores your selection.

> **Note**
>
> While the inline diff is displayed, the editor shows syntax errors from the diff rendering. These errors clear after you accept or reject the change.

Spec-driven development with AWS Coding Plugin for SAP ABAP

Spec-driven development (SDD) makes a written specification, not the code, the primary artifact. A *spec* is a set of documents that state what to build. The plug-in generates the SAP ABAP from the spec, so the code is an output rather than the source of truth. SDD is a discipline, not a feature of any one tool, and AWS Coding Plugin for SAP ABAP ("Coding Plugin") brings it to Eclipse and SAP ABAP Development Tools (ADT).

Why spec-driven development matters for SAP ABAP

SAP ABAP code is not disposable. It lives for decades, passing through many hands as teams, owners, and organizations change. A spec keeps a durable record of intent that outlives any one developer.

- The spec-writing process mirrors how ABAP development is conducted, with an analysis of business requirements before designing and coding a solution.
- The spec is a living document, readable by a future developer, human or agent.
- Later changes start from stated intent, not from reverse-engineering the code.
- The requirements defined in the spec are testable by Coding Plugin and become part of the self-correcting loop that ensures it produces accurate business logic.
- The spec-writing process is the primary collaboration between human and agent. While writing the spec, both the human and agent learn and explore the business requirements, discover edge cases, and plan for an optimal result.

The three spec documents

Kiro records a spec as three documents. Each builds on the one before it.

- `requirements.md` states what to build: the object type, inputs and outputs, business rules, error cases, which patterns to use or avoid, and the security requirements.
- `design.md` states how to build it: the technical design and the tables, CDS views, APIs, and other objects it depends on. Coding Plugin queries the ABAP system to understand the available objects in the system during this stage.
- `tasks.md` breaks the design into an ordered list of implementation tasks that the agent executes. The task list can branch into parallel tasks where ABAP objects can be modified by multiple subagents without conflict.

Specs are stored in the `.kiro/specs/<feature>/` directory of your project, so they can be committed to a durable Git repository for sharing with the team.

Working with specs in Eclipse

Coding Plugin supports the spec workflow in the UI. Certain keywords such as "spec" will automatically activate spec mode, or you can manually set a chat tab to spec mode in the mode drop-down. In spec mode, you will see spec action buttons at the top of the chat tab at certain points in the spec workflow.

These buttons appear according to which spec documents exist on disk.

- **Generate Design** appears while the spec has requirements but no design.
- **Generate Tasks** appears once a design exists.
- **Run All Tasks** appears when the task list exists. It opens a new tab and reports one line for each task as the status changes (queued, in progress, completed, failed, or stopped).

When the agent needs a decision during the workflow, it pauses and renders the choices as clickable option cards. When a prompt looks like it should be a spec, the plug-in nudges you to start one.

As it executes tasks, the agent verifies its own work through the self-correcting loop (see the self-correcting loop).

Coding Plugin estimates credit usage in developing each spec, which is useful to customers and service providers to evaluate a project's credit consumption (see Monitoring credit usage). For what a credit is, see Pricing.

How specs are written

The spec-writing process is collaborative. You and Kiro define the requirements, design the solution, and develop a task list. During the design phase, Coding Plugin benefits from a live SAP connection so that it can discover the tables, APIs, and objects your system actually offers.

Managing change and code reviews

AWS Coding Plugin for SAP ABAP ("Coding Plugin") gives Kiro the tools to assign your changes to transports and to work with your SAP system's Change and Transport System. It adds no version history of its own. You review and revert the changes it makes with your SAP system's own version control, exactly as you would review and revert your own.

Working with transports

Coding Plugin gives Kiro tools to check whether an object needs a transport, list and search your transports, create a transport, add objects to it, and release it. Coding Plugin's default steering guides it to ask before creating a transport, and to ask first whether you want to reuse an existing one — such as one created by Change Request Management (ChaRM). You should provide additional steering on how your organization handles transports.

Avoid commingling human-written and agentically written code in the same transport. This facilitates a code review by the developer after a spec has been implemented.

Note the following limits:

- Older ABAP systems do not offer ADT tools to release transports. On these systems Coding Plugin works around this by invoking SAP GUI to present SAP's standard release dialog instead.
- As with all tool calls, creating or releasing a transport is subject to Kiro's permissions gates. You can deny Kiro permission to release a transport (see Kiro Permissions).

Reviewing and reverting changes

Coding Plugin and Kiro have no built-in code review process. To review and undo the changes Coding Plugin made, use your SAP system's own version control, which gives you two options.

Native ABAP version control

SAP's Change and Transport System (CTS) manages how changes move between SAP systems. Its native version control records a version of an object when its transport is released. You can compare an object to a previous version and revert to it. It works through transports, so it does not cover local ($TMP) objects.

To use it effectively:

1. Release your previous transports before the Coding Plugin begins its work, so your current code has a recorded version to compare to.
2. Ask Kiro to keep its work in a single transport, so the transport collects the changes to compare and revert.
3. After the work is released, use ADT functions to view the object history and compare the changed objects to their previous versions. For code objects, the Coding Plugin can read past versions and help you compare them to the current version.
4. If you need to revert, use ABAP version control to restore the previous version.

abapGit

abapGit gives you Git-controlled version history and diff on a per-package basis. Push entire packages to a Git repository such as AWS CodeCommit, make your changes with the plug-in, and diff the result with abapGit.

The Coding Plugin Model Context Protocol (MCP) server supports certain abapGit operations when they are available as ADT tools, enabling Kiro to add and pull repositories for you. ADT tools for abapGit are built into BTP ABAP.

Best practices

These practices help you get the most from Coding Plugin. Agentic development for SAP ABAP changes frequently. For the latest guidance and innovations, see the AWS for SAP blog.

Keep each initiative in its own tab. Open a new tab for each effort you undertake, and do not mix separate efforts in one tab. A tab holds one conversation, so a focused tab keeps Kiro's context window on that one effort. For spec-driven development, author the spec in one tab and implement it in another, so each tab's context can focus on a single phase.

Keep agentic work in its own transport. Do not mix your own changes and the changes Coding Plugin makes in the same transport. Release any open transport before you start, so those changes land in a fresh one. A dedicated transport makes it clear what changed, and lets you compare versions and revert changes with SAP's version management tools.

Keep scratch objects out of transports. Tell Kiro to create temporary objects and test harnesses in `$TMP`, or another local ($) package, so they are never transported to production. Put this instruction in a steering file so Kiro applies it in every session.

Review changes before you release. When you finish, ask Kiro to walk you through its changes. The Coding Plugin can open code editors to show you code, and can read past versions to help you understand what changed.

Customizing AWS Coding Plugin for SAP ABAP

You customize AWS Coding Plugin for SAP ABAP ("Coding Plugin") in two ways: through steering, which shapes what Kiro knows about your organization and your SAP systems, and through the plug-in's own Eclipse preferences.

Steering

Steering gives Kiro durable, project-level instructions that apply to every prompt, kept as Markdown files under `.kiro/steering/`. For how steering works and how to write your own, see Steering in the Kiro documentation.

Coding Plugin ships built-in ABAP steering and adds two commands that generate steering for your environment. All of it is plain Markdown in your project, so you can review and edit any of it.

Built-in ABAP steering

Coding Plugin ships steering that teaches Kiro how ABAP development works, including the self-correcting loop (see ABAP development). This steering is active by default and needs no setup.

Organization steering

Run `/workspace-init` to record your organization's standards as steering: its development goals, naming conventions, clean core approach, and change-management practices. The command runs automatically the first time you open a workspace, and you can run it again to update what it wrote.

Per-system steering

Run `/sap-discover` to set up a connected SAP system. The command runs automatically when you connect to a new SAP system. The agent interviews you about the system and writes your answers to `.kiro/steering/sap-system-<SID>.md`. This steering gives Kiro context about the system's role (Development, QA, and so on), important packages, useful test data, and other details that help Kiro develop effectively.

Writing your own steering

You can add your own steering files under `.kiro/steering/`. For example, if your team has naming conventions, a preferred error-handling pattern, or packages that should never be

modified, write those rules as a Markdown file and Kiro follows them in every session. Kiro can write steering for you. For how to write effective steering, see Steering in the Kiro documentation.

AWS Coding Plugin for SAP ABAP preferences reference

AWS Coding Plugin for SAP ABAP ("Coding Plugin") adds its own preferences to Eclipse, separate from Kiro's configuration.

AWS Coding Plugin for SAP ABAP preferences

Open the preferences from **Preferences**, AWS Coding Plugin for SAP ABAP. The page you land on holds the connection to Kiro and to SAP. The rest of the settings are grouped by task on four subpages. Eclipse lists them in this order: **Onboarding**, **Behavior**, **MCP Servers**, and **Diagnostics**.

Main page

- **Kiro CLI location** shows the status of the `kiro-cli` binary and holds the path to it. Edit the path directly, choose **Browse** to pick the file, or choose **Test** to check the path you entered. A blank path means Coding Plugin detects `kiro-cli` on your PATH.
- **Log in to Kiro** starts the Kiro CLI authentication flow, so you can sign in without leaving Eclipse.
- **ADT Status** reports whether SAP ABAP Development Tools (ADT) is installed, and links to the installation if it is not.

Onboarding

- **Show Welcome on startup** opens the Welcome wizard the next time Eclipse starts.
- The **Onboarding nudges** checkboxes each suppress one of the one-time offers: to take the guided tour of the chat panel, to set up steering for the workspace, or to set up steering for a connected SAP system. The SAP setting applies to every SAP system you connect to.
- **Run Welcome Wizard** opens the onboarding wizard now, without restarting Eclipse.

Behavior

- **Terminal command timeout (seconds)** is how long a command may run before Coding Plugin stops it. The timeout is not a gate on whether a command runs at all: the agent asks you to approve each command first, and does not run it if you decline. See Kiro Permissions.

- **Session/prompt timeout (seconds)** is how long a prompt may run before it times out.
- **Recent conversations to list** is how many past conversations the chat panel's recent-conversations selector offers, newest first. Set it to `0` to list every conversation the agent remembers.
- **Warning threshold (%)** and **Critical threshold (%)** are percentages of the context window. When usage crosses the warning threshold, Kiro compacts the conversation automatically. Crossing the critical threshold escalates the warning and fires a safety-net compaction if one is not already in progress. For more about the context window, see Using the chat.
- **Detect spec-building prompts and offer to switch to Spec mode** offers to switch to Spec mode when a prompt looks like it is describing a feature to build.

MCP Servers

This page covers your Model Context Protocol (MCP) servers, and which of the plug-in's own tools the agent can reach.

- **User MCP servers** and **Workspace MCP servers** give the path of each `mcp.json` file, with a link that opens it, or creates it from a starter template if it does not exist yet. The workspace row becomes available once a Coding Plugin project is open. Kiro reads these files itself, so an entry you add takes effect on your next session.
- **Bridge Tool Categories** selects which categories of the plug-in's own tools the agent can see and call: **Coding Plugin Introspection**, **Eclipse tools**, and **Additional SAP ADT functions**. Clearing a category hides its tools from the agent. Changes take effect after a session restart.

Diagnostics

Use these settings to capture logs when you report a problem.

- **Log verbosity level** sets how much detail the plug-in records: `ERROR`, `WARN`, `INFO`, or `DEBUG`.
- **Capture KRS request payloads (requires restart)** and **Capture ACP frames** record the full model requests, and the Agent Client Protocol (ACP) messages exchanged with Kiro CLI. Both are available only at the `DEBUG` verbosity level. **Remove chat content from captured frames**, available while frame capture is on, replaces prompts, replies, and file contents in the frame log with a size marker.
- **Collect Diagnostics** bundles the logs, your chat transcripts, and the plug-in's configuration into a single archive you can attach to a support case.

- **Record usage metrics in the log** writes each turn's token counts, credit usage, duration, and context usage to the log file on your own machine. Nothing is sent off your machine.
- **Read `mcp.json` files for diagnostics** gives Coding Plugin read-only access to your user-level and workspace-level `mcp.json` files. The read serves diagnostics alone: it warns you when one of your entries targets SAP's built-in ADT MCP endpoint, and it records an entry count in a diagnostics archive. Kiro reads and honors your `mcp.json` files regardless of this setting; see Architecture in How AWS Coding Plugin for SAP ABAP works.

> **Warning**
>
> At the DEBUG verbosity level, and when **Capture ACP frames** is on, the captured logs can include your prompts and chat transcripts. A diagnostics archive also bundles your chat transcripts. If you share an archive without reviewing it first, you can expose the contents of your prompts and transcripts to whoever receives it.

For when and how to collect diagnostics while troubleshooting, see Troubleshooting.

Theme, fonts, and colors

The chat panel follows Eclipse's own appearance settings, so it matches the rest of your IDE and changes live when you adjust them.

- **Theme.** The chat panel supports both light and dark themes. Choose a theme under **Preferences**, **General**, **Appearance**. The panel repaints to match.
- **Fonts and text size.** Set the chat text under **Preferences**, **General**, **Appearance**, **Colors and Fonts**, in the AWS Coding Plugin category. **Content Font** sets the transcript and input text (the size of headings scales with it). **Chrome Font** sets the tab strip, buttons, and status areas.
- **Colors.** The same AWS Coding Plugin category lets you override the chat card, code block, and accent colors. If you do not override them, the panel derives its colors from your current theme.

Updates and maintenance

You update AWS Coding Plugin for SAP ABAP ("Coding Plugin") through Eclipse, from the same update site you installed it from. The plug-in and Kiro CLI update on two independent tracks: Eclipse updates the plug-in when you run Help > Check for Updates, and Kiro CLI updates itself separately. Updating one does not update the other.

Check for plug-in updates

Eclipse checks the update sites already configured for your installation, including the Coding Plugin update site, and offers any newer version it finds.

To update Coding Plugin

1. In Eclipse, choose **Help**, **Check for Updates**.
2. If a newer version is available, choose AWS Coding Plugin for SAP ABAP in the **Available Updates** list, then choose **Next**.
3. Review the update details, then choose **Next**.
4. Accept the license terms, then choose **Finish**.
5. When Eclipse prompts you to restart, choose **Restart Now**.

After Eclipse restarts, the new version of the plug-in is running.

Review release notes

When the plug-in's version changes, the What's New window opens automatically after Eclipse restarts and summarizes what changed in that release.

Uninstall AWS Coding Plugin for SAP ABAP

Uninstalling removes the plug-in from Eclipse. Your SAP system, your ABAP Development Tools (ADT) connection, and Kiro CLI are not affected.

To uninstall Coding Plugin

1. Choose **Help**, **About Eclipse** (macOS: **Eclipse**, **About Eclipse**).

2. Choose **Installation Details**.
3. On the **Installed Software** tab, choose AWS Coding Plugin for SAP ABAP.
4. Choose **Uninstall**.
5. Follow the prompts to finish, then restart Eclipse when prompted.

After Eclipse restarts, the plug-in's chat window and editor integration no longer appear. To install the plug-in again, choose **Help**, **Install New Software** and enter the update site `https://downloads.aws.com/sap/aws-coding-plugin-for-sap-abap/current`.

Security

AWS Coding Plugin for SAP ABAP ("Coding Plugin") runs on your machine, inside your own Eclipse installation. Its security model is a set of trust boundaries between the parts it connects. These boundaries are: the plug-in and Eclipse, the plug-in and Kiro, Kiro and the Model Context Protocol (MCP) servers, and the path to your SAP system. This chapter describes each boundary and what crosses it.

Topics

- The plug-in and Eclipse
- The plug-in and Kiro
- Kiro and the MCP servers
- Reaching your SAP system
- Credential and token handling

The plug-in and Eclipse

Coding Plugin is an OSGi bundle that runs inside your Eclipse installation. It runs with the permissions of your Eclipse session and can read and modify files in your Eclipse workspace. It holds no permission that you do not already have as the developer running Eclipse.

The plug-in and Kiro

On your machine, Coding Plugin launches Kiro CLI as a child process and communicates with it over a local stdio pipe. Communication between Coding Plugin and Kiro does not cross the network.

Kiro CLI handles sign-in to the backend service, using any sign-in option that Kiro supports, such as AWS Builder ID or AWS IAM Identity Center. For more information about sign-in options, see Authentication on the Kiro website.

What data leaves your machine

Your prompts, and the code context the agent gathers, are sent by Kiro CLI to the Kiro backend service, which runs the model. How that data is handled is governed by Kiro. See Privacy and security in the Kiro documentation. Coding Plugin adds no separate data collection of its own.

Kiro and the MCP servers

Kiro CLI connects to MCP servers configured in `mcp.json`.

The two ADT-related MCP servers (the SAP ADT MCP server, and Coding Plugin's MCP server) are a special case:

The Coding Plugin MCP server is reachable only by your Kiro CLI:

- It binds to `localhost` (`127.0.0.1`) only and is never exposed on the network.
- It listens on an ephemeral port chosen when it starts.
- Its URL paths are randomized.
- Access requires a bearer token that is regenerated each time the server starts.

The connection from Kiro to the MCP server is configured through `stdio`, outside of `mcp.json`, so that the bearer token is never written to persistent storage.

Coding Plugin also automatically configures the connection from Kiro to SAP's ADT MCP server using the `stdio` channel instead of `mcp.json`. The plug-in never writes SAP's bearer token to persistent storage.

Reaching your SAP system

The agent reaches your SAP system only through the SAP ADT connection you have already defined in Eclipse. This connection is authenticated to SAP by the SAP ADT plugin, and Coding Plugin never gains access to these credentials. The plug-in stores no SAP credentials of its own.

Every read and write runs as your SAP user. The agent can do nothing on the system that your user cannot do, and SAP's own authority checks apply to its actions exactly as they apply to yours. Your SAP authorizations such as `S_DEVELOP` and the transport layer remain the fundamental control.

Developers often have powerful permissions in the development system, and so does Coding Plugin. You can constrain the agent's actions through steering and prompts. For example, your steering files can contain instructions such as "never modify code in the `Z_HCM` package". However, steering is not a security measure. A large language model (LLM) can ignore steering because of context rot, conflicting instructions, or misunderstanding it. Your SAP authorizations remain the fundamental control.

Every tool call the agent makes is gated by Kiro Permissions, Kiro's mechanism for approving the agent's actions. See Kiro Permissions.

Credential and token handling

- **Kiro sign-in** is managed by Kiro CLI. The plug-in does not have access to Kiro's credentials to the backend service.
- **Your SAP logon** is your existing ADT connection. The plug-in does not have access to your SAP credentials.
- **SAP ADT's MCP server token**: SAP mints it and persists it in its own Eclipse preference store. Coding Plugin reads it from there and keeps no copy of its own.
- **The plug-in's MCP server bearer token**: Coding Plugin holds it in memory, regenerates it each time the server starts, and never persists it.

Troubleshooting AWS Coding Plugin for SAP ABAP

Most problems with AWS Coding Plugin for SAP ABAP ("Coding Plugin") surface as a banner in the chat panel that names the problem and offers an action. The following list covers the problems you are most likely to encounter, most common first.

Common issues

The agent cannot reach your SAP ADT MCP server

You enable SAP's ABAP Development Tools (ADT) Model Context Protocol (MCP) server under **Preferences**, **ABAP Development**, **MCP Server**. The server requires a port and a token, both of which the preferences page generates when you enable it.

The most common causes are:

- The ADT MCP server is not running. Toggle it off and on.
- No token is defined. Open the preferences page and enable the server again to generate one.

An MCP server failed to connect

When a session starts, the plug-in connects the agent's MCP servers. If one does not answer, the agent runs without it, and a banner lists the servers that failed.

- A server can exceed Kiro's MCP launch timeout.
- A failure of the plug-in's own server is a problem in the plug-in.
- A failure of a server you defined in `mcp.json` points at that entry.

  In the banner, choose **Reconnect** for a single failed server or **Reconnect all** when several failed. A reconnect can take up to a minute.

Kiro CLI is missing, out of date, or signed out

The plug-in launches `kiro-cli` to run Kiro. If `kiro-cli` is not installed, is older than the version the plug-in requires, or you are signed out, the plug-in shows a **Fix setup…** banner. Choose it to open the setup wizard at the step that is not satisfied.

- If `kiro-cli` is not installed, install it from Kiro CLI on the Kiro website, then choose **Check Again**.
- If it is out of date, update it and choose **Check Again**.

- If you are signed out, follow the wizard to sign in.

A chat stops responding

A chat can stop because the agent is still working on a long turn, is waiting for you to answer a question or approve a tool call, the connection to `kiro-cli` has dropped, or the plug-in itself is at fault. The plug-in reports which of these it found.

- If the agent is still working, let it run, or stop it.
- If the agent is waiting on you, answer the question or approve or deny the tool call.
- If Kiro exits, the plug-in attempts to restart it (the tab shows **Reconnecting...**).
- A stopped turn shows **Response stopped**, with **Resend** and **Continue**.

Additional help

If the agent is responsive, ask it to diagnose itself. In many cases it can report its own state and read its own logs, and offer an explanation of an error.

If the agent cannot chat with you, a common cause is an expired Kiro login or license. Try running `kiro-cli` directly from the command line to confirm whether the underlying Kiro tool is working.

If you cannot resolve an issue, open a technical support case in the AWS Support Center and attach a diagnostics bundle. Choose **Kiro** as the service and **IDE** as the category. For instructions on collecting a bundle, see Collecting diagnostics for a support request.

Logging

AWS Coding Plugin for SAP ABAP writes detailed diagnostic information to a log file on your local machine. You set the log verbosity level in **Preferences**, AWS Coding Plugin for SAP ABAP, **Diagnostics**.

What is recorded at INFO level

At the default INFO level, the main log (`~/.aws_cp4sa/aws_cp4sa.log`) records major lifecycle events: startup steps, the Kiro CLI path and version, the MCP bridge starting and its port, SAP ADT MCP discovery, session create, load, and close events, connection state changes and the reason for any restart, the names of the built-in MCP servers offered to each session, and model selection. It

records no chat content — neither your prompts nor the agent's replies. Two entries do reflect your work: the full path of any file or folder you pin into chat context, and a size estimate, in bytes, of each request sent to the agent.

What is recorded at DEBUG level

DEBUG level records the same information as the INFO level, and the following additional diagnostic trace: every path checked while locating the CLI, the command used to start it, every protocol message by name and size, every file read, file write, and shell command the agent performs together with your approval decision, every ADT tool call and its timing, hook evaluation including your configured file patterns, and the resolved workspace and project paths. Large values are recorded as a length rather than their contents, so prompt and file text still does not appear. The log level propagates to the Kiro CLI, which writes its own log, `kiro-cli.log`.

Additional logging settings

Three more checkboxes appear on the **Diagnostics** page when the log verbosity level is DEBUG.

- **Capture KRS request payloads** (off by default) — Makes the Kiro CLI write the full model-request payloads, the prompts and context actually sent to the model, to `~/.aws_cp4sa/krs-dumps/`, from its next start.
- **Capture ACP frames** (off by default) — Writes every Agent Client Protocol message exchanged with the CLI, one JSON object per line, to `~/.aws_cp4sa/acp-frames.log`, taking effect on the next message.
- **Remove chat content from captured frames** (on by default, available only while capture is enabled) — Replaces prompts, replies, file contents, and tool-call titles in that file with a size marker, keeping paths, tool names, and MCP server names. Clearing it records that content as written.

Collecting diagnostics for a support request

When you contact support, attach a diagnostics bundle.

1. Open **Preferences**, AWS Coding Plugin for SAP ABAP, **Diagnostics**.
2. Choose **Collect Diagnostics**. The plug-in writes a ZIP file.
3. Attach the ZIP file to your support request.

The ZIP file contains the plug-in's logs, your configuration, a snapshot of the plug-in's state, and the result of its self-diagnosis.

> **Note**
>
> The bundle includes your chat transcripts (prompts and replies). At the DEBUG log level with **Capture ACP frames** on, raw messages exchanged with Kiro CLI are also captured. Secrets such as tokens are stripped automatically from ACP frames. Review an archive before you share it.

Pricing

AWS Coding Plugin for SAP ABAP is offered at no cost. You pay only for your consumption of Kiro credits. For plans, included credits, and overage rates, see Pricing in the Kiro documentation.

AWS Coding Plugin for SAP ABAP FAQ

Common questions about AWS Coding Plugin for SAP ABAP ("Coding Plugin"). Each answer links to the chapter that covers the topic in full.

Getting started

What does the plug-in cost, and what is a credit?

There is no charge for the plug-in itself. The Kiro work you do through it draws *credits* from your Kiro plan, where a credit is the unit Kiro meters its work in. See Pricing.

Can I use my own AI model?

No. Kiro runs the model, and the model and mode you can choose come from what your Kiro backend offers, shown in the chat panel's selector bar. You cannot point the plug-in at an arbitrary external model. See The selector bar.

How do I ask the agent to write code that matches our conventions?

Create steering describing your conventions. If your team already has a document or webpage describing your conventions, have Kiro read it and extract it into steering. The `/workspace-init` skill will interview you about this. See Customizing Coding Plugin.

Can I use other Kiro features, such as powers and hooks?

Coding Plugin surfaces many Kiro capabilities in the Eclipse UI, including steering, skills, specs, MCP servers, Kiro Permissions, and the model and mode selector. Kiro capabilities that have no dedicated UI in the plug-in, such as hooks, are configured through files under the `.kiro/` folder in your project. For what Kiro offers and how to configure it, see the Kiro documentation on the Kiro website.

Working with SAP

What SAP ABAP objects can the agent create and change?

The agent can work with any SAP ABAP object that ADT exposes. This includes classes, programs, function modules, Core Data Services (CDS) views, DDIC objects, and ABAP RESTful Application

Programming Model (RAP) artifacts. Capabilities vary depending on the ABAP system version. See Working with ABAP.

Do I need SAP's ADT MCP server?

Recommended, not required. The plug-in's own ABAP Development Tools (ADT) tools reach SAP through your ADT connection independently of SAP's Model Context Protocol (MCP) server, but SAP's server alone offers Joule AI fixes for ABAP Test Cockpit (ATC) findings and Open Data Protocol (OData) service discovery. See Architecture.

Can I use Coding Plugin for languages other than SAP ABAP?

Coding Plugin is an SAP ABAP specialist persona and is not suited to other languages. Development for non-ABAP scenarios, such as the SAP Cloud Application Programming Model (CAP), is better suited to SAP-provided tools such as SAP Joule for Developers.

Can I read and implement SAP Notes with Coding Plugin?

The SAP ADT plug-in cannot implement Notes the way SNOTE does, so the Coding Plugin cannot either.

SAP ADT cannot read Note text. However, you can download Notes as PDF documents and ask Kiro to read them from your Downloads directory, or attach them to your prompt. In this way, Coding Plugin can analyze short dumps and compare them to symptoms documented in specific Notes you download.

Can I perform Basis tasks with Coding Plugin?

Coding Plugin can read short dumps and gateway errors to help diagnose certain kinds of Basis errors, but it is not a general-purpose Basis tool.

SAP ADT does not expose Basis tasks such as scheduling background jobs or editing RFC destinations, so Coding Plugin cannot perform these tasks.

How does the plug-in use temporary programs?

When writing conventional code such as Java and Python, Kiro knows how to create temporary programs to test unknowns such as library behavior. This technique works in ABAP too, where Kiro can create a temporary executable ABAP class to test the behavior of a BAPI or other API that it is unfamiliar with. We recommend steering Kiro to put these in $TMP or another $ package so

that they are not transported to production, and to limit itself to read-only operations unless you explicitly approve the test of a write operation.

How can I get Kiro to implement my spec faster?

The speed of agentic processing depends largely on the inference latency of the LLM. Larger models reason more deeply but take longer to process. Lighter models can process much faster and perform well when executing a well-defined task list (for example, implementing a spec). Kiro supports running up to five concurrent subagents, so when developing a spec, ask Kiro to emphasize parallelism in the task sequencing.

How can I get Kiro to write better SAP ABAP?

Unlike SAP Joule, the underlying Kiro engine does not have the benefit of a purpose-built model trained specifically in ABAP and SAP knowledge. But today's commodity LLMs show strong ABAP knowledge, and Kiro can look up live information from the web. To add your own expertise on SAP ABAP or specific SAP APIs, create a steering document such as `sap-abap.md` and add your knowledge there.

Can Kiro write SAP ABAP code that calls AWS services?

Yes. With the AWS SDK for SAP ABAP, ABAP programs can call more than 240 AWS services in pure SAP ABAP. To give Kiro detailed knowledge of the SDK's modules and APIs, add the AWS SDK for SAP ABAP Knowledge MCP server to your `mcp.json` file. For setup, see Using the AWS SDK for SAP ABAP Knowledge MCP Server in the *AWS SDK for SAP ABAP Developer Guide*. For more about the SDK, see What is AWS SDK for SAP ABAP?.

Security and data

Does my SAP ABAP code leave my machine?

Yes. Your prompts, and the code context the agent gathers, are sent by Kiro CLI to the Kiro backend service, which runs the model. For how Kiro handles your data, see Privacy and security in the Kiro documentation and What data leaves your machine.

How do I limit what the agent can do in my SAP system?

The agent runs as your SAP user, so your SAP authorizations are the real limit, and Kiro Permissions gates every action behind your approval. See Reaching your SAP system and Kiro Permissions.

Can I use third-party steering or MCP servers for SAP ABAP?

AWS does not make recommendations on third-party or open source tooling. Subject any third-party steering or agentic tooling to a security review before you use it. For more information, see MCP security in the Kiro documentation and Agentic AI security in the AWS Prescriptive Guidance.

Document history

The following table describes important changes to the AWS Coding Plugin for SAP ABAP Developer Guide.

Change	Description	Date
Initial release	Initial release of the AWS Coding Plugin for SAP ABAP Developer Guide.	October 26, 2026