Custom actions: passing the source record to your own codeunit

Advanced Workflow can run **your own codeunit** at three points in a workflow's life: when the workflow completes, when it is cancelled with an action, and when an escalation step fires. In each case the app can hand your codeunit the **source record** that the workflow is running on — the purchase order, the customer, the item, whatever the template's primary table is.

This page is the developer contract: what exactly is passed, how to receive it, and what the app does with the result.

Where a custom codeunit can be configured

Trigger point	Where you configure it	Setting that controls the handover
Workflow completed	**Workflow Template** card, section **Workflow Action** → **Action on Completion** = **Custom action**	**Pass Source Record**
Workflow cancelled with action	**Workflow Template** card, section **Cancel Custom Action Configuration** → **Cancel Action** = **Custom action**	**Cancel Pass Source Record**
Escalation step reached	**Escalation Step** card → escalation action **Custom action**	**Pass Source Record**

In all three places you choose **Custom Action Object Type** = **Codeunit** and enter the object ID in **Custom Action Object ID**. The lookup is filtered to codeunits, so you can pick from the list rather than typing the number.

> The same three settings also accept **Report** as the object type. A report is rendered to PDF instead of being run; see *Passing the record to a report* at the end of this page.

What exactly is passed

When **Pass Source Record** is enabled, the app calls your codeunit with the source record attached, which in AL terms means your codeunit's `OnRun` trigger receives that record in `Rec`. When the setting is disabled, the codeunit is run **without** any record — `Rec` is not populated.

Two properties of the handover matter in practice:

- **Exactly one record is passed, and no filters are set on it.** Your codeunit receives the single source document. If your logic needs the related lines, read them yourself from the header's key fields.
- **The record is the source document the workflow was started on, identified by its record ID.** Before running a completion action the app refreshes the record from the database if it has unsaved changes in memory, so your code sees the stored state rather than a half-modified copy. If your logic depends on a field that other processes may have changed in the meantime, it costs nothing to re-read the record with `Get` at the start of your `OnRun` and be certain.

Writing the codeunit

Declare `TableNo` as the table your workflow template runs on. That is what makes `Rec` a typed record you can address by field name:

```
codeunit 50100 "MY Release Approved PO"
{
    TableNo = "Purchase Header";

    trigger OnRun()
    var
        MySetup: Record "MY Integration Setup";
    begin
        // Rec is the purchase order the workflow was running on.
        Rec.TestField("Document Type", Rec."Document Type"::Order);

        MySetup.Get();
        if not MySetup."Notify ERP Gateway" then
            exit;

        SendToGateway(Rec."No.", Rec."Buy-from Vendor No.", Rec.Amount);
    end;
```

```
    local procedure SendToGateway(DocumentNo: Code[20]; VendorNo: Code[20]; DocAmount:
Decimal)
    begin
        // your integration call
    end;
}
```

Point the template's **Custom Action Object ID** at `50100`, keep **Pass Source Record** enabled, and the codeunit runs every time a workflow on **Purchase Header** completes.

Modifying the source record

You may change and save the source record inside your codeunit. The app suppresses its own trigger detection while the action runs, so your `Modify` does not start a second workflow on the same document:

```
    trigger OnRun()
    begin
        Rec."Your Custom Status" := Rec."Your Custom Status"::Approved;
        Rec."Approved On" := CurrentDateTime();
        Rec.Modify(true);
    end;
```

> If all you need is to set a few fields to fixed values, you do not need a codeunit at all — use **Define field value assignments** as the action instead and configure the fields in the **Field Value Assignments** section.

Serving several tables

Because `TableNo` is fixed per codeunit, one codeunit fits the templates of one table. When the same logic should serve templates on several tables, write a thin codeunit per table and keep the logic in one shared codeunit that they all call:

```
codeunit 50101 "MY Workflow Action - Purch"
{
    TableNo = "Purchase Header";

    trigger OnRun()
    var
```

```
        SharedLogic: Codeunit "MY Workflow Action Logic";
    begin
        SharedLogic.NotifyGateway(Rec.RecordId(), Rec."No.", Rec.Amount);
    end;
}

codeunit 50102 "MY Workflow Action - Sales"
{
    TableNo = "Sales Header";

    trigger OnRun()
    var
        SharedLogic: Codeunit "MY Workflow Action Logic";
    begin
        SharedLogic.NotifyGateway(Rec.RecordId(), Rec."No.", Rec.Amount);
    end;
}
```

Each template then points at the codeunit matching its own primary table.

> Give the codeunit the `TableNo` of the template's primary table. If the two do not match, the record cannot be handed over and the run is written to the audit trail as failed. A codeunit with no `TableNo` at all is only appropriate together with **Pass Source Record** switched off.

Error handling and the audit trail

Every run is logged to the workflow's audit trail, which you find on the workflow instance under **Find workflow instances**:

- On success — **Custom codeunit 50100 executed successfully**
- On failure — **Custom codeunit 50100 execution failed**
- If the object type or ID is missing — **Custom action not configured properly**

> An error inside your codeunit does **not** roll back the workflow and does not stop the document from being completed. The workflow stays completed, the failure is written to the audit trail, and processing continues. If your integration must not be lost silently, make your codeunit record its own failure — write a queue entry or a log record you monitor — rather than relying on the error surfacing to the user.

For the same reason, do not put a `Message` or a `Confirm` in a custom action. The action often runs in a background session — on escalation it runs from a job queue, and on completion it may be deferred to a background task — where there is no user to answer a dialog.

Passing the record to a report

With **Custom Action Object Type** = **Report** the app does not run a report request page. It renders the report to PDF in the background, and with **Pass Source Record** enabled it filters the report's dataset down to that one source record before rendering. Your report therefore needs a `DataItem` on the source table for the filter to apply.

> The generated PDF is not stored automatically. If you need the document archived, use a codeunit that produces and saves it, or enable **Save Audit Trail to Attachments** on the template for the workflow's own audit PDF.

Related pages

- **Workflow action on completion** — configuring the action in the template
- **Module: Escalation Engine** — escalation steps and their actions
- **Find workflow instances** — where to read the audit trail

---

Revision #1
Created 2026-08-18 10:44:21 UTC by Bernd Feddersen
Updated 2026-08-18 10:44:22 UTC by Bernd Feddersen