CogniDev Workbench

IBM i (AS/400) adaptive playbook

DATA SHEET · SOURCE COVERAGE AND TARGET MAPPING

This sheet covers the IBM i (AS/400) adaptive playbook: what it detects in a source export, how it runs, what it produces, and where each AS/400 artifact lands on a .NET, Java or other target.

1 What the Workbench is

The Workbench is a development environment for engineering work on existing and new systems: modernization, decomposition, migration, governance and new builds. It reads the codebase first and then runs the work as a playbook. It runs on your machine, and no source leaves it.

2 How it works: playbooks

A playbook is the unit of work. It is a versioned, signed process that sets out the steps in order, the points where you approve, the tools it may run, the standards it holds the work to, and the questions it is allowed to ask. None of that is decided while it runs, so the same input produces the same process every time.

IBM i is served by **one adaptive playbook** rather than one playbook per scenario. It reads the source first, then shapes its remaining steps around what it found and the target you choose. A rehost, a rewrite and an API-enablement share the same reading steps and differ only where they have to.

3 The playbook pipeline

The stages below run in order. Each one is labeled with what drives it. **Structural** means parsers and generators, which produce the same result every time. **AI** means a model is called, always against facts already established from the source. **Hybrid** means the model proposes and structure or tools decide.

	STAGE	DRIVEN BY	WHAT HAPPENS
1	**Structural intelligence**	Structural	The source is parsed and joined into one model of the system: programs, files, screens, batch jobs and the calls between them. No model is involved, so nothing here is inferred.
2	**AI planning**	AI	What the system does in business terms, the target design, and the order of work. Written down before anything changes and put to you for approval.
3	**Task generation**	Hybrid	The approved plan becomes an ordered list of tasks, one file at a time. The breakdown comes from the structural model; the model fills in intent and sequence.
4	**Scaffolding**	Structural	Project structure, wiring, configuration and boilerplate are generated from tested blocks selected by your answers. The same answers produce the same tree.
5	**Agentic loop**	AI	Code is generated and validated task by task. Each task is a small change, a real build, then a commit. A failing task stays on that task rather than stacking a second change on the first.
6	**Testing**	Hybrid	Tests and test data are generated from the model and the schema, then run by the test tooling. The tools decide whether something passes; the model is never asked whether the code works.
7	**Final validation and next steps**	Hybrid	Whether the result behaves the way the source did, unit by unit. Compilers, tests and checks do the proving; the model reconciles the results, explains what could not be proved, and states what remains.

The rule the playbook follows throughout is that anything derivable from the source is derived, never asked. The model is the finishing layer over the structural read, not the engine underneath it.

4 Source coverage

The playbook detects and parses the following member types. Extensions are matched without regard to case, and the steps that run are decided by which of these are actually present.

CATEGORY	MEMBER TYPES	WHAT IS READ FROM IT
RPG	`.rpg` · `.rpgle` · `.sqlrpg` · `.sqlrpgle` · `.rpginc` · `.rpgleinc` · `.rpg36` · `.rpg38` · `.rpt` · `.rpt36` · `.rpt38`	Program name and dialect, file declarations with their device and usage, record-level reads and writes, embedded SQL, subroutines and procedures, program calls, display formats, Open Access handlers, commit and rollback points.
CL	`.clp` · `.clle` · `.clp38` · `.cld`	Program calls, submitted jobs, file overrides, data area and data queue use, file commands, message sends, commitment control scoping.
DDS	`.pf` · `.lf` · `.dspf` · `.prtf` · `.dds` · `.pf38` · `.lf38` · `.dspf36` · `.dspf38` · `.prtf38` · `.print38` · `.icff` · `.menu`	Physical files as tables with typed columns and keys, logical files as access paths over a parent file, display files as screen formats with their fields, subfile pairs and function keys, printer files as report layouts.
COBOL/400	`.cblle` · `.sqlcblle` · `.sqlcbl` · `.cbl36` · `.cbl38` · `.cicscbl` · `.cicscblle` · `.cicssqlcbl`	The IBM i bindings only: files assigned to the database, DDS record layouts copied into the program, record-level I/O, and calls. General COBOL structure is read by the COBOL parser.
SQL members	`.table` · `.sqltable` · `.view` · `.sqlview` · `.index` · `.sqlindex` · `.alias` · `.sqlalias` · `.sqlprc` · `.sqludf` · `.sqludt` · `.sqlseq` · `.sqltrg`	Tables, views, indexes and aliases as schema. Stored procedures, functions, types and sequences as routines with their parameters. Used where a shop has moved off DDS.
Commands	`command definition source (.cmd38, or by content)`	Command name, the program it runs, and its declared parameters with types and defaults.
Panel groups	`.pnlgrp` · `.uim`	UIM help topics joined to the command parameter they document, menus, and display panels.
Binder source	`.bnd` · `.bnddir`	The procedures a service program exports, and the binding directories that group them. This is the declared module boundary.
System objects	`.msgf` · `.dtaara` · `.dtaq`	Message descriptions with their text, help and severity. Data area and data queue definitions with declared type and size.
Triggers	`.systrg` · `.sqltrg` · `.trg`	The table, the event, the timing, and the program that runs. Covers both the ADDPFTRG command form and the SQL CREATE TRIGGER form.
Scripting	`.rexx` · `.rex` · `.ocl` · `.ocl36` · `.txt38`	The CL commands and program calls each script issues. REXX and System/36 OCL both drive the system by handing it CL.
ILE C	`.c` · `.h (IBM i bindings only)`	Record-level I/O through the C record API, DDS structures brought in by #pragma mapinc, and CL commands issued from C.
Host bridges	`Java (JT400)` · `Node.js and Python (XMLSERVICE / itoolkit)`	Calls from off-platform code into programs, commands, files and data queues on the box. Where the target is chosen at run time, the resource kind is recorded instead of a name.
Build metadata	`Rules.mk` · `iproj.json` · `.ibmi.json`	The declared object dependency graph, the library list that resolves unqualified names, the include path, and the declared encoding of the source.

Encoding. Source exported from EBCDIC is frequently not valid UTF-8. Files that fail a UTF-8 read are decoded one byte per character, which preserves the fixed column positions that RPG and DDS depend on.

5 How members are identified

A source export is rarely tidy. Members are commonly renamed, and a shop's own suffixes carry no standard meaning. The playbook uses three methods, in this order.

METHOD	APPLIES TO	NOTES
Extension	All member types listed in section 4	The mapping from extension to language is held in one configuration file. Adding a member type is a configuration change, not a code change.
Content	Renamed members, and command definition source	Fixed-format members are recognized by the form indicator in column 6 behind the five-column sequence area. CL is recognized by its statement shape. Command source has no reliable extension, because .cmd is a Windows batch file elsewhere.
Parser confirmation	Anything proposed by content	Content detection is deliberately permissive so that a renamed member is found. The parser makes the final decision, and a member is only recorded as RPG if it parses as RPG.

Historical copies. Members whose name marks them as a prior version of a live member are matched to that member and flagged. Renamed members with no live counterpart are counted as live, because nothing in the source states that they are retired.

6 What the playbook produces

The parsed members are joined into a single model of the system. The playbook delivers it in the sections below, each one traceable back to the member and line it came from.

FACT	DESCRIPTION
Program inventory	Every program with its language and dialect, its source member, and the objects it declares.
Call graph	Program-to-program calls, including calls issued from CL, REXX, OCL, ILE C and from off-platform code. Merged into the same graph used for every other language.
Data model	Physical files and SQL tables as typed entities with columns, lengths and keys. Logical files recorded as access paths over their parent.
Data access	Every record-level read and write, every embedded SQL statement, and every CL file command, located at file and line. Each is classified against the device its target was declared on, so a screen write is never recorded as a database write.
Screen inventory	Every display format with its fields, the subfile control and detail pairs that make a scrolling list, and the function keys.
Runtime overrides	Every OVRDBF and OVRPRTF, which rebind a file at run time so the file a program opens is not the file it names.
Batch and shared state	Submitted jobs, data area use, data queue use, and the CL commands issued from code.
Transaction boundaries	Commit and rollback operations and commitment control scoping.
Triggers	Table, event, timing and the program that runs, for both trigger forms.
Field cross-reference	Which programs reference which column, bounded by the files each program declares. Supports impact questions at field level rather than file level.
Message surface	Every message description with its text, help text and severity, keyed by the identifier programs raise.
Module boundaries	Service program exports from binder source, and the binding directories that group them.
Declared dependencies	The object dependency graph, library list and encoding taken from build metadata where the shop maintains it.
Missing objects	Every reference to a file, record format or screen that the export does not contain, reported by name with the program and line that reaches for it.

FACT	DESCRIPTION
Navigation views	Entry points, reachability, layers, domains, most-changed members, and end-to-end flow and sequence views over the call graph.

Every section can be read in the Workbench, exported for a report, or handed to another tool. The playbook is deterministic: two runs over the same export produce the same result.

7 Source to target mapping

An AS/400 combines the language, the database, the screens, the batch scheduler and the packaging into one system. A migration has to place all five, so the playbook works in artifacts rather than in files. The table below is the target for each AS/400 artifact. The playbook holds the mapping; the target column it uses is the one you select when the plan is made.

AS/400 ARTIFACT	.NET	JAVA	ANY OTHER TARGET	NOTES
RPG program, batch	C# class in a service project	Java class in a Spring service	Service class or module	Business logic moves as a unit. The program cycle and indicators are replaced by explicit control flow.
RPG program, interactive	API controller plus a web page	REST controller plus a web page	HTTP endpoint plus a UI view	Screen handling and business logic are separated. In the original they are the same program.
RPG subprocedure, ILE module	Method or class	Method or class	Function or method	Already a callable unit with a signature, so this is the cleanest mapping in the set.
Service program and binder source	Class library or project boundary	Module or build artifact	Module or package boundary	The export list is a declared public surface and is the natural seam for a decomposition.
CL program	Hosted service or console job	Spring Batch job or shell script	Job or workflow script	CL is orchestration. It maps to job code, not to business logic.
Submitted job (SBMJOB)	Scheduled job on Hangfire or Quartz	Scheduled job on Quartz or Spring Scheduler	Scheduled job	Job dependencies come from the CL call graph.
DDS physical file (PF)	SQL table plus EF Core entity	SQL table plus JPA entity	Relational table plus a mapped type	Column types and lengths carry across directly. Packed decimal maps to decimal, not to floating point.
DDS logical file (LF)	Index, or view where it selects or joins	Index, or view where it selects or joins	Index or view	A keyed logical becomes an index. A select or omit logical becomes a view.
Record-level I/O	EF Core or Dapper repository method	Spring Data or JDBC repository method	Repository or data-access method	CHAIN, READ, SETLL, WRITE and UPDATE become set-based or keyed queries. Cursor semantics have to be preserved where a program reads in order.
Embedded SQL	Parameterized query	Parameterized query	Parameterized query	Closest mapping of all. The main work is dialect differences and host variable binding.
DDS display file and subfile	REST endpoint plus a paged grid	REST endpoint plus a paged grid	Paged list endpoint plus a UI view	A subfile control and detail pair is a server-paged list. Function keys become named actions.
DDS printer file (PRTF)	Reporting component or generated PDF	Reporting component or generated PDF	Report or document generator	Column positions in the printer file define the layout.
Data area	Configuration store or a table row	Configuration store or a table row	Configuration store or table row	Frequently used as a counter or a last-used key. Concurrency behavior has to be preserved.

AS/400 ARTIFACT	.NET	JAVA	ANY OTHER TARGET	NOTES
Data queue	Service Bus, RabbitMQ or Kafka	RabbitMQ, Kafka or JMS	Message queue or stream	Keyed data queues need a queue that supports selective receive.
Message file (MSGF)	Resource file	Resource bundle	Localized message catalog	The user-facing message text lives only here and has to be carried across.
Trigger	Database trigger, or a domain event	Database trigger, or a domain event	Database trigger or domain event	Decide per trigger. Keeping it in the database preserves behavior; moving it into the application makes it visible.
Commitment control	Transaction scope	Declarative transaction	Transaction boundary	The boundary has to match the original exactly, including which files are under commit.
Command (CMD)	CLI command or admin endpoint	CLI command or admin endpoint	Command-line or admin operation	Parameter types and defaults are declared in the command source.
File override (OVRDBF)	Runtime configuration or a repository parameter	Runtime configuration or a repository parameter	Runtime configuration	Overrides are why a static read of the program alone gives the wrong file.
REXX and OCL scripts	Shell script or pipeline step	Shell script or pipeline step	Script or pipeline step	These issue CL, so they follow wherever the CL goes.
JT400 or XMLSERVICE bridge	Direct in-process call	Direct in-process call	Direct in-process call	Once the called program has moved, the bridge is removed rather than reimplemented.

Database target

DB2 for i maps to SQL Server, Azure SQL, PostgreSQL or Oracle, or it can be left in place with the application moved first. The data model is produced from DDS and SQL DDL in the same shape either way, so the database decision does not have to be made before the assessment.

Any other target

.NET and Java are the two targets in normal use for this platform, so they are named in full. The third column is the shape each artifact takes on any other runtime, and the Workbench builds and migrates to Node and TypeScript, Go, Rust, Python and Scala on the same mapping with the framework names substituted. Because the playbook adapts, the target is chosen when the plan is made rather than fixed in advance, so the decision can be made after the source has been read.

8 Engagement types this supports

Customers mean different things by moving off the AS/400. The playbook reads the source the same way for all of them and then adapts, so the engagement type is an answer it takes rather than a different playbook.

TYPE	WHAT IT NEEDS FROM THE ASSESSMENT
Rehost	Object inventory and build dependencies. The code does not change.
Refactor in place	Which programs are already off the green screen, and where the module boundaries are.
Rewrite on a new stack	The full model, including screens, triggers and transaction boundaries.
Replace with a package	The data model and the business rules, to compare against what the package covers.
Data offload	The data model and which programs write to each file.
API enablement	Service program exports and the call graph, to decide what to expose.
Knowledge capture	The field cross-reference and the message surface, before the people who know the system leave.

9 Verification

The playbook's parsers were built and checked against source exports from working IBM i shops. Between them those exports cover fixed-format RPG III, modern free-form RPG, ILE service programs and ILE C, COBOL/400 in a single-byte codepage, a shop that keeps its schema as SQL DDL rather than DDS, and a shop that has already moved off the green screen.

Results are repeatable. Two runs of the playbook over the same export produce the same result, so a figure quoted in an assessment can be reproduced.