



UltraGX manual

Installing, licensing and using UltraGX: from the download to a documented KB ready to migrate.

Version 0.2.8 · ultra-gx.com

Contents

01 Introduction

02 Installation

03 First start and licence

04 The application

05 Documenting a KB

06 Reading the result

07 AI narrative

08 User notes

09 Migrating with an AI agent

10 Command line

11 Troubleshooting and FAQ

12 Glossary

Introduction

UltraGX reads a GeneXus Knowledge Base (KB), from version 7.5 to 18, and puts everything it learns about it into open files: Markdown documents to read, JSON to process and a browsable website that works offline. The result serves three purposes:

- **Understanding the system** without opening the GeneXus IDE: what each object does, which business rules it applies, which tables it reads and writes, who calls it and what it calls.
- **Preserving the knowledge** in a format you can version, search and share, even when the people who built the system are gone.
- **Migrating**: the output is designed so that an AI agent (Claude Code, Codex, Cursor, Antigravity) can regenerate the application in another technology, module by module, without interpreting GeneXus from scratch again.

What it does, step by step

1. **Reads the KB** from an XPZ/XML export, from a text export or, on Windows with GeneXus installed, by opening the KB directly with the connector.
2. **Analyses each object** with deterministic rules: type, structure, rules, events, variables, navigation, data access and references. No language model guesses at this stage: every conclusion is tied to the evidence that produced it.
3. **Relates the objects**: it builds the dependency graph, the data model, the call trees, the data flow diagrams and the *clusters* (functional groups of objects that work together).
4. **Writes the documentation**: one document per object, the data model, the clusters, a rebuild roadmap and the web viewer.
5. **Optionally, adds AI prose** (the *narrative*): a language model writes explanations and the engine checks that every name it cites exists in the object before publishing them.

Read-only

UltraGX **never modifies the KB**. The connector opens the Knowledge Base read-only and has no write path at all; with an XPZ or a text export you do not even need GeneXus installed. Everything it generates is written to a separate output folder.

Editions

	Demo	Full
Duration	10 days from installation	per the licence
Inventory, graph, data model, clusters, call trees and diagrams	the whole KB	the whole KB
Detailed document per object (rules, pseudocode, implicit SQL, interface, original code)	25 objects chosen by the engine	all

	Demo	Full
Knowledge packs, rebuild plans per cluster, AI narrative	sample only	all
Internet connection	required on every run (licence check)	not needed

In the demo, the remaining objects get a summary document with their identity, location, profile, confidence, cluster and references, plus a notice that the detail is available in the full edition.

What data leaves your machine

- **The KB analysis runs entirely on your computer.** The code, rules and structure of your objects are not sent to any UltraGX server.
- **Demo licence:** when you register you send your contact details; on every run the licence service receives the licence identifier, the machine id, **how many** objects the KB has (counts only) and the **names** of the objects proposed for the sample.
- **Full licence:** there is no communication with the service after activation.
- **AI narrative:** if you use it, each object's document is sent to the provider you choose (Anthropic, OpenAI, Google, etc.) with your own API key. If you would rather nothing leaves your network, use a local model (Ollama or LM Studio). See [AI narrative](#).

How to read this manual

- **If you are trying the demo:** [Installation](#) → [First start and licence](#) → [Documenting a KB](#) → [Reading the result](#).
- **If you already have a licence:** [Documenting a KB](#) → [AI narrative](#) → [User notes](#).
- **If you are part of the migration team:** [Reading the result](#) → [Migrating with an AI agent](#) → [Command line](#).

Installation

All downloads are at ultra-gx.com. Each file is published with its SHA-256 fingerprint, in case you want to check that the download arrived intact.

Requirements

System	Minimum version	What else it needs
Windows	Windows 10 version 1809 or Windows 11, 64-bit	The Microsoft Edge WebView2 runtime (the installer adds it if missing)
Linux	Debian 12, Ubuntu 22.04, Fedora 39 or equivalent, 64-bit	<code>libwebkit2gtk-4.1</code> (or 4.0) and GTK 3
macOS	macOS 13, Apple Silicon or Intel	Nothing

You do not need to install Python, Node or .NET: the engine and the application come compiled.

To read a KB live (without exporting it first) you need Windows with GeneXus X Evolution 1 to 18 installed on the same machine. For any other case an XPZ/XML export or a text export is enough, and then the operating system does not matter.

Disk space: a 700-object KB produces about 100 MB of documentation; a 5,000-object one, around 1 GB.

Windows

With the installer (recommended)

1. Download `UltraGX-<version>-win-x64-setup.exe`.
2. Run it. If Windows shows "**Windows protected your PC**" (SmartScreen), click **More info** and then **Run anyway**. The warning appears because the installer is still new to SmartScreen, not because there is a problem with the file.
3. **Language:** choose English, Español or Português. That language is used for the wizard, for the licence agreement shown and as the initial language of the application and of the documents it generates. Both can be changed later.
4. **Licence agreement:** read it and accept it to continue.
5. **Destination folder:** by default `C:\Users\<user>\AppData\Local\Programs\UltraGX`. The installation is per user and does not ask for administrator rights.
6. **Additional tasks** (both optional):
 - *Create a desktop shortcut.*
 - *Open GeneXus exports (.xpz) with UltraGX:* with this option, double-clicking an `.xpz` opens UltraGX on the **Run pipeline** tab, ready to document it.
7. **Install.** If the computer lacks WebView2, the installer downloads and installs it in this step (an internet connection is required).
8. When it finishes, leave **Launch UltraGX** ticked to open it and continue with [First start and licence](#).

On Windows the installer also includes the **GeneXus connector**, which lets you read a KB live. If GeneXus is not installed on the machine, the connector stays inactive and does no harm.

Linux

Choose the format that matches your distribution:

```
# Debian / Ubuntu
sudo apt install ./ultragx-<version>-amd64.deb

# Fedora / RHEL
sudo dnf install ./ultragx-<version>-1.x86_64.rpm
```

The packages install the application in `/opt/ultragx`, the `ultragx` command and the entry in the applications menu, and they declare the WebKitGTK dependencies, so the package manager installs them for you.

AppImage (no installation): make it executable and open it. In this case the dependencies must already be present on the system:

```
sudo apt install libwebkit2gtk-4.1-0      # Debian / Ubuntu
chmod +x UltraGX-<version>-x86_64.AppImage
./UltraGX-<version>-x86_64.AppImage
```

On Linux the "live KB" source is not available (the connector needs GeneXus, which runs on Windows): export the KB to XPZ or to text on the machine where GeneXus is and document it here.

macOS

1. Download the `.dmg` that matches your Mac: `osx-arm64` for Apple Silicon (M1 onwards), `osx-x64` for Intel.
2. Open it and drag **UltraGX** to **Applications**.
3. The macOS build is not signed with an Apple Developer ID, so macOS blocks it the first time. Double-click **Allow UltraGX.command** in the disk image (or run `xattr -cr /Applications/UltraGX.app` in Terminal), then open UltraGX. If macOS still objects, go to **System Settings → Privacy & Security** and click **Open Anyway** next to the UltraGX message.

As on Linux, on macOS you work with XPZ or text exports.

Updating

Download the new version and install it over the previous one (on Windows, the installer replaces the files in the same folder). Settings, narrative connections, the licence and the generated outputs are kept.

Outputs generated with an earlier version can still be opened. To benefit from the improvements of the new engine, run the pipeline again on the same KB: user notes and the narrative already generated are kept.

Uninstalling

- **Windows:** *Settings* → *Apps* → *Installed apps* → *UltraGX* → *Uninstall*.
- **Linux:** `sudo apt remove ultragx` or `sudo dnf remove ultragx`; for the Applmage, just delete the file.
- **macOS:** drag UltraGX from Applications to the Bin.

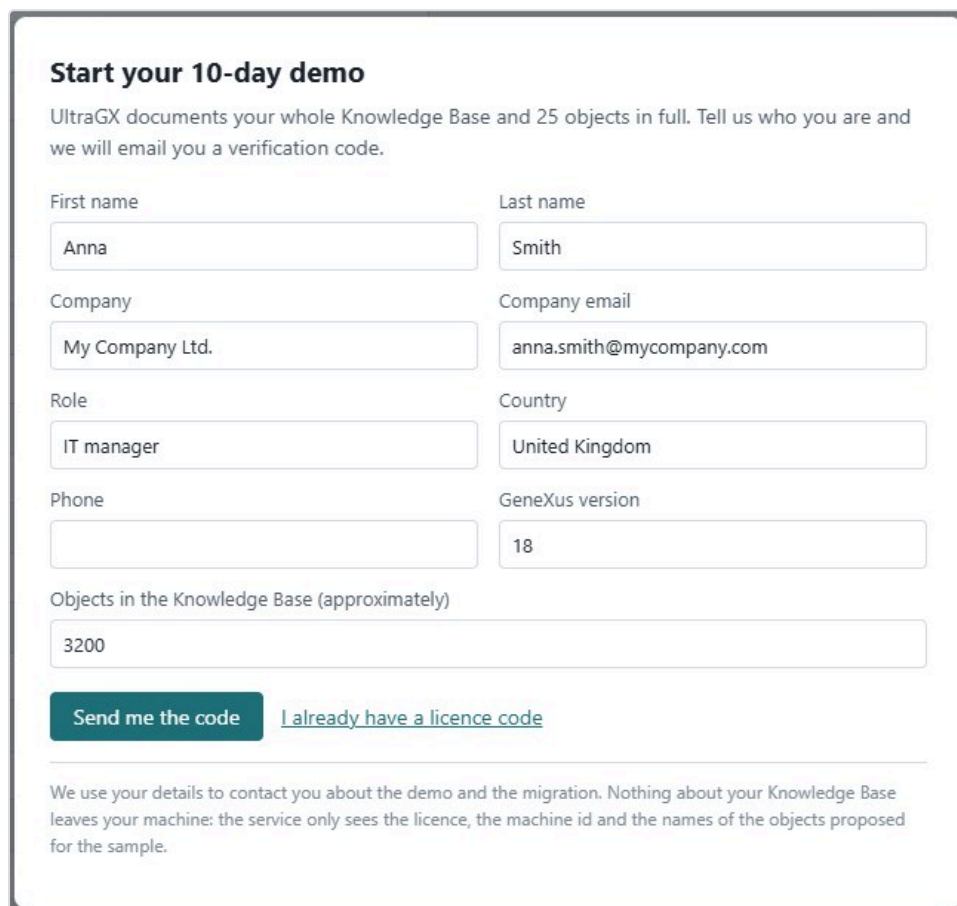
Uninstalling **does not delete** your data:

What	Where
Settings, narrative connections (including API keys) and licence	%APPDATA%\UltraGX on Windows, ~/.config/UltraGX on Linux and macOS
Generated outputs	the output folder you chose
Demo period record	kept on purpose: reinstalling does not restart the 10 days

If you want to remove everything, also delete the settings folder and your output folder. Remember that the settings store your API keys.

First start and licence

UltraGX needs a licence to run the engine. The first time you open the application without a licence, the **Start your 10-day demo** form appears. From there you can request the demo or, if you have already bought the full edition, use your code.



The screenshot shows a web form titled "Start your 10-day demo". Below the title is a paragraph: "UltraGX documents your whole Knowledge Base and 25 objects in full. Tell us who you are and we will email you a verification code." The form contains several input fields arranged in two columns. The left column has fields for "First name" (containing "Anna"), "Company" (containing "My Company Ltd."), "Role" (containing "IT manager"), "Phone" (empty), and "Objects in the Knowledge Base (approximately)" (containing "3200"). The right column has fields for "Last name" (containing "Smith"), "Company email" (containing "anna.smith@mycompany.com"), "Country" (containing "United Kingdom"), and "GeneXus version" (containing "18"). At the bottom left is a dark green button labeled "Send me the code". To its right is a link that says "I already have a licence code". Below the form fields is a small disclaimer: "We use your details to contact you about the demo and the migration. Nothing about your Knowledge Base leaves your machine: the service only sees the licence, the machine id and the names of the objects proposed for the sample."

Demo registration form

Requesting the demo

1. Fill in the form. **First name, Last name, Company** and **Company email** are required; the rest (**Role, Country, Phone, GeneXus version, Objects in the Knowledge Base (approximately)**) helps us prepare the conversation about the migration.
2. Use your **company email**. The demo is issued to companies evaluating a migration and the service rejects personal email addresses (Gmail, Outlook, Hotmail, Yahoo, iCloud and similar).
3. Click **Send me the code**. Within a couple of minutes an email arrives with a six-digit code.
4. Type the code and click **Activate the demo**. The code is valid for **15 minutes**.

Start your 10-day demo

UltraGX documents your whole Knowledge Base and 25 objects in full. Tell us who you are and we will email you a verification code.

We sent a six-digit code to anna.smith@mycompany.com. It is valid for 15 minutes.

4 8 2 9 1 3

Activate the demo

[Back](#)

No email? Check the spam folder, or go back and try another address.

We use your details to contact you about the demo and the migration. Nothing about your Knowledge Base leaves your machine: the service only sees the licence, the machine id and the names of the objects proposed for the sample.

Verification code step

If the email does not arrive, check your spam folder. If it still does not show up, click **Back** and request the code again (or try another company address).

Once the demo is active, the form closes and the application shows the **DEMO edition** notice in the top bar.

What the demo includes

- **10 days** counted from installation on that machine. Reinstalling or requesting another code does not restart the period.
- One demo per machine and one per email address.
- The whole KB is analysed and documented at the level of inventory, data model, graph, clusters, call trees and diagrams. **25 objects** get the detailed document; the engine chooses them among the most connected business objects, spread across types, so that the sample is representative. The same 25 are kept even if you run the KB several times.
- Every run **needs internet** to check with the licence service. If there is no connection, the run still finishes but without detailed objects, and the notice is shown in the README and in the viewer.

Activating the full edition

When you buy the licence you will receive a code of the form `ULGX-XXXX...`.

- **On first start:** in the form click **I already have a licence code**, paste the code and click **Activate with a code**.
- **With the application already in use** (for example, when moving from the demo to the full edition): go to **Settings → License**, paste the code into **Activation code** and click **Activate with a code**. Leave the **License service URL** field empty unless we tell you otherwise.

Activation binds the licence to that machine. After activation, the full edition does not connect to the service again.

Installing a licence file

For installations without internet access we can give you an `ultragx.lic` file instead of a code:

1. In **Settings → License** copy **This machine id** and send it to us.
2. When you receive the file, click **Install license file...** and select it.

The file is stored in `%APPDATA%\UltraGX\ultragx.lic` (Windows) or `~/.config/UltraGX/ultragx.lic` (Linux and macOS).

License

Status	valid
Edition	full
Owner	My Company Ltd. <it@mycompany.com>
Expires	2036-09-04
License file	%APPDATA%\UltraGX\ultragx.lic
This machine id	3f9c2a71b8e4d05c (send it to the owner to get a license bound to this machine)

Activation code (DEMO-XXXX / ULGX-XX)

License service URL (optional)

Activate with a code

Install license file...

Settings, License block

Checking the licence status

Settings → License shows:

Field	What it means
Status	valid , or the reason why it is not
Edition	DEMO with the number of detailed objects and the service it checks with, or full
Owner	company and email it was issued to
Expires	expiry date, or never
License file	where it is installed
This machine id	identifier used to bind the licence

Registration and activation messages

Message	Cause	What to do
Please use your company email address...	You entered a personal email address	Register with your company address
This email address already activated a demo...	That address has already been used	Write to contact@ultra-gx.com if you need more time

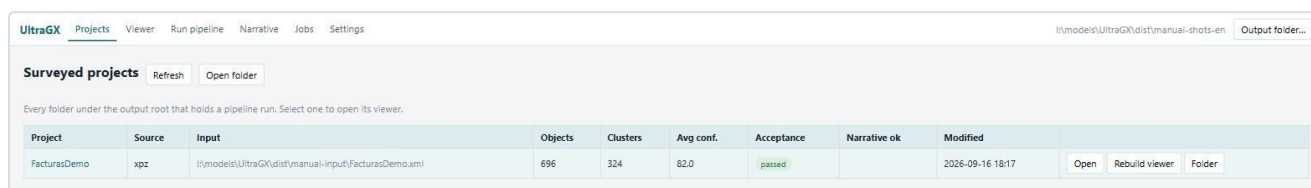
Message	Cause	What to do
The demo period on this machine is over...	The 10 days have passed	Write to us to continue with the full edition
Wrong code...	The code does not match	Check the latest email received and type it again
The code expired...	More than 15 minutes have passed	Back and request a new one
Too many wrong codes...	Five failed attempts	Back and start again
Too many attempts from this network...	Many requests in a row from the same network	Wait a while and try again
Check the fields...	A required field is missing or not valid	Fill in first name, last name, company and a valid email
The licence service could not be reached...	No internet, or a proxy/firewall blocks <code>licenses.ultra-gx.com</code>	Check the connection; on corporate networks ask for HTTPS to <code>licenses.ultra-gx.com</code> to be allowed

The application

The UltraGX window has a top bar with six tabs and, on the right, the active **output root** with the **Output folder...** button to change it. Each pipeline run creates a sub-folder inside that folder; it is best to choose one on a disk with free space and outside synchronised folders (OneDrive, Dropbox), because a large KB generates thousands of files.

The first time, if no output folder is set, the application opens **Settings** directly.

Projects



Projects tab

Lists every folder under the output root that contains a pipeline run:

Column	What it shows
Project	name of the sub-folder
Source / Input	where the KB was read from (<code>xpz</code> , <code>gxtext</code> , <code>live</code>) and the path
Objects / Clusters	number of documented objects and of functional groups
Avg conf.	average confidence of the analysis, from 0 to 100
Acceptance	result of the final quality check (<code>passed</code> , <code>warning</code> , <code>failed</code>)
Narrative ok	objects with published AI prose
Modified	date of the last run

Open takes the project to the **Viewer**; **Rebuild viewer** rebuilds its website; **Folder** opens it in the file manager. **Refresh** reads the output folder again and **Open folder** opens the whole output folder.

Viewer

Shows the website generated for the project chosen in the drop-down list. It is explained in detail in [Reading the result](#).

- **Reload**: reloads the page (useful while the narrative is running, since it keeps updating the documents).
- **Rebuild viewer**: rebuilds the site from the output files, without analysing the KB again.
- **Open in browser**: opens the same viewer in your browser. Everything can be read from the browser, but user notes can only be added from the application.

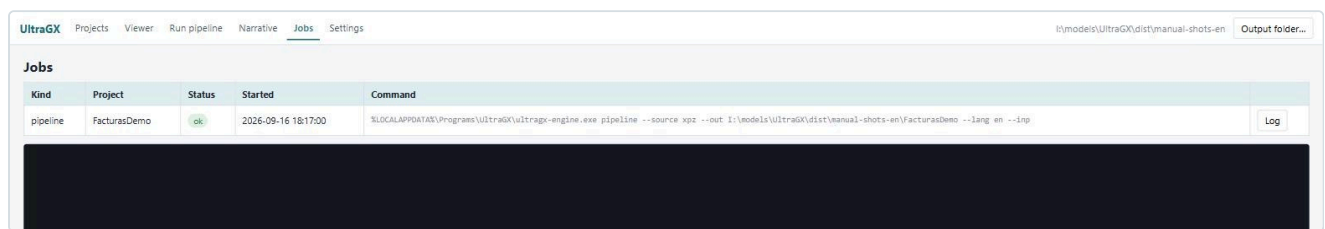
Run pipeline

This is where a KB is documented: you choose the source, the input, the project name and the language of the documents. See [Documenting a KB](#).

Narrative

Adds prose written by a language model to the documents of an already generated project, with connections to different providers. See [AI narrative](#).

Jobs



Jobs tab

Every pipeline and every narrative you launch becomes a job with its **Kind**, **Status**, **Started** and **Command**. **Log** shows the full output of the job (also for finished ones, as long as the application stays open) and **Stop** cancels it. You can run one pipeline and one narrative at the same time, but not two pipelines or two narratives simultaneously.

Do not close the application with jobs in progress: tracking and the log are lost. Wait for them to finish or stop them first. A stopped narrative can be **resumed** later without repeating the objects already processed.

Settings

UltraGX

Projects

Viewer

Run pipeline

Narrative

Jobs

Settings

Settings

Interface language

English

Output root (one sub-folder per run)

It\models\UltraGX\dist>manual-shots-en

Browse...

Save saved

License

Status valid

Edition full

Owner My Company Ltd. <it@mycompany.com>

Expires 2036-09-04

License file %APPDATA%\UltraGX\ultragx.lic

This machine id 3f9c2a71b8e4d05c (send it to the owner to get a license bound to this machine)

Activation code (DEMO-XXXX / ULGX-XX) License service URL (optional) Activate with a code

Install license file...

Environment

App 0.2.7 · Microsoft Windows 10.0.26100

Engine compiled: %LOCALAPPDATA%\Programs\UltraGX\ultragx-engine.exe (UltraGX engine 0.2.7 (pack rel-2026.09))

GeneXus connector %LOCALAPPDATA%\Programs\UltraGX\connector\UltraGX.Gx.Gateway.exe

GeneXus installed 18u5 (18.0.5.44509), 18 (18.0.15.57673), 18u10 (18.0.10.53188), 17 (17.0.11.32605), 16 (16.0.11.15999), 15 (15.0.4.48249), XEv3 (10.3.15.50288), XEv2 (10.2.7.25097)

Output root ok

Settings file %APPDATA%\UltraGX\settings.json (or ~/.config/UltraGX)

Settings tab

- **Interface language:** English, Español or Português (BR). It does not change the language of the documents, which is chosen on each run.
- **Output root:** the same one shown in the top bar. Click **Save** after changing it.
- **License:** status, activation with a code and installation of a licence file. See [First start and licence](#).
- **Environment:** what the application detected on the machine.

Environment	What it means
App	version and operating system
Engine	path and version of the compiled engine in use
GeneXus connector	path of the connector, or not found (always on Linux and macOS; on Windows only when using a copy without the connector)
GeneXus installed	GeneXus installations found, with their version; if there are none, you work with XPZ or text exports
Output root	ok if it exists
Settings file	where settings, connections and the licence are stored

Documenting a KB

Documenting a Knowledge Base means running the **pipeline** on it: the **Run pipeline** tab reads the KB, analyses it and writes the complete documentation into a sub-folder of the output folder.

The screenshot shows the 'Run the pipeline' tab in the UltraGX application. The interface includes a top navigation bar with tabs: UltraGX, Projects, Viewer, Run pipeline (active), Narrative, Jobs, and Settings. Below the navigation bar, the 'Run the pipeline' section contains several input fields and a 'Run' button. The 'Source' dropdown is set to 'xpz — XPZ / XML export (GeneXus 7.5 to 18)'. Below this, a text box explains that it reads XPZ or XML exports of GeneXus 7.5, 8, and 9 (the old KMW 2.x format) and of X Evolution 1 to 18. The 'Input' field contains the path 'I:\models\UltraGX\dist\manual-input\FacturasDemo.xml' with a 'Browse...' button. The 'Project name (output sub-folder)' field contains 'FacturasDemo'. The 'Language' dropdown is set to 'English'. There is a checkbox for 'gxtxt: skip referenced modules (ref/)' which is currently unchecked. A green 'Run' button is at the bottom left of the form. Below the button is a link for 'Recent inputs'.

Run pipeline tab

1. Choosing the source

Source	What you choose in Input	When to use it
xpz — XPZ / XML export (GeneXus 7.5 to 18)	an <code>.xpz</code> or <code>.xml</code> file	The general case. It works for any version and on any operating system. The GeneXus version is detected automatically
gxtxt — text export folder (src/ + ref/)	the folder that contains <code>src/</code> and <code>ref/</code>	If you already have a GeneXus 17 or 18 text export
live — open the KB with the GeneXus installed on this machine (connector)	the KB folder (the one that contains the <code>.gxw</code>)	Windows with GeneXus X Evolution 1 to 18 installed. Nothing needs to be exported by hand
mcp — raw JSON dumps folder	a folder of JSON dumps	Internal use and support; normally not used

Below the selector, the application explains each source. If the **live** option appears disabled (*needs the connector and a GeneXus installation*), the machine has no GeneXus or you are using a copy

without the connector.

How to get an XPZ

In the GeneXus IDE: **Knowledge Manager → Export**, select all the objects of the KB and export to an `.xpz` file. In old versions (7.5, 8, 9) the equivalent path is **Knowledge Manager → Distribute**. Include everything: the more complete the export, the better the references, the data model and the clusters.

Tips:

- Export from the up-to-date **development KB**, not from an old copy.
- If the KB is very large and the IDE cannot export it in one go, export it in parts (by module or by object type) and document each file as a separate project, or use the **live** source.
- If you documented the KB with a version of UltraGX earlier than 0.2.5 and the SDTs appear without members, run the pipeline again (and, with the live source, the export) with the current version.

Live source and GeneXus version

With **live**, UltraGX opens the KB with the installed GeneXus, exports it to text inside the project folder (`_source_text/`) and then runs the pipeline. In **GeneXus version for the live KB** leave **auto (from the KB)**: the connector picks the installation that matches the version the KB was saved with. If you have several installations of the same version (for example two *upgrades* of GeneXus 18), you can force one.

Before running with **live**:

- **Close the KB in the GeneXus IDE.** A KB can only be open in one process at a time; if the IDE (or another tool) has it open, the export waits or fails.
- Expect the live export to be the slowest stage: the connector extracts each object through the GeneXus SDK. A 5,000-object KB can take more than an hour just to export. The log shows which object is being extracted.

2. Project name and language

- **Project name (output sub-folder):** by default, the name of the input file or folder. If you run again with the same name, the project is regenerated in the same folder and the **user notes** and the **narrative** already generated are kept.
- **Language:** English, Español or Português (BR). It is the language of the document prose and of the viewer. The `README.md` at the root of the output is always in English, because it is meant for AI agents.
- **gxtext: skip referenced modules (ref/):** with the gxtext and live sources, leaves out the objects of modules that the KB only references (GeneXus and third-party libraries). It speeds up the run and cleans up the inventory; the downside is that calls to those modules remain as external references.

3. Running

Click **Run**. The log below shows progress by stage:

```

[ 2.7s] analysing 576/696 WebPanel.GAMMAuthenticationTypeEntry
[ 3.7s] analysing 600/696 WebPanel.GAMMUserPermissions
[ 4.7s] analysing 625/696 PatternSettings.WorkWithPlus
[ 4.8s] analysing 696/696 WorkWithPlusInstance.WorkWithPlusFactura
[ 4.8s] extracted 696 objects
[ 4.8s] semantic rules and validation on 696 objects
[ 5.6s] indexing the KB
[ 5.6s] resolving references and building the graph
[ 5.8s] clustering
[ 5.9s] rebuild plan per cluster
[ 5.9s] cluster knowledge and functional flow
[ 6.5s] writing 696 object documents
[ 6.5s] data model (foreign keys and ERDs)
[ 6.6s] writing 1/696 Procedure.GoogleDirectionsServiceRequest
[ 7.6s] writing 256/696 Domain.GAMTimeZone
[ 8.6s] writing 457/696 Procedure.WWP_ItemInCollection
[ 9.6s] writing 587/696 WebPanel.GAMSecurityPolicyEntry

```

Log of a run

- **Reading:** the engine reads the source and names the object it is processing.
- **Analysis:** rules, events, variables, data access and references of each object.
- **Graph, clusters, knowledge and rebuild plan:** relationships between objects.
- **Writing:** documents, data model, diagrams and viewer.
- **Acceptance:** final quality check.

To avoid flooding the screen, the log writes at most one line per second per stage. If a line repeats with the same object for a while, the engine is still working on a large object: it has not hung.

When it finishes successfully, the status changes to **done**, the project appears in **Projects** and opens automatically in the **Viewer**. If the run fails, the status says **finished with status error** and the log shows the reason (see [Troubleshooting](#)).

Reference times

KB	Source	Approximate time
700 objects	XPZ	less than a minute
5,000 objects	XPZ or text export	from a few minutes to half an hour
5,000 objects	live	the export (an hour or more) plus the pipeline

Times depend on the disk and on the size of each object's code. With the demo, a brief check with the licence service is added.

Recent inputs

Recent inputs keeps the last fifteen inputs used. Click one to load it into **Input** again.

Opening an XPZ with a double click

If you ticked *Open GeneXus exports (.xpz) with UltraGX* during installation, double-clicking an `.xpz` opens the application on **Run pipeline** with the **xpz** source and the file already loaded: you only need to check the project name and the language, and click **Run**.

Reading the result

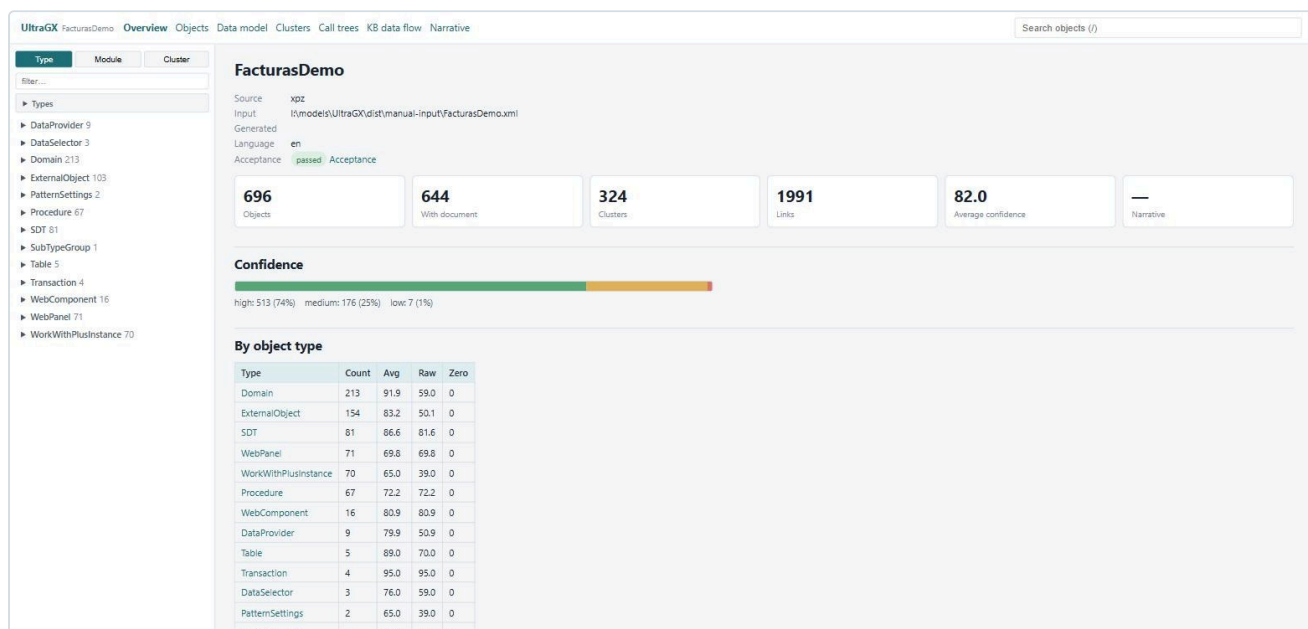
Each run leaves two things: a **web viewer** to read and browse, and a **folder of files** in Markdown and JSON to process, version or hand to an AI agent. Both have the same content.

The viewer

The viewer opens in the application's **Viewer** tab or with **Open in browser**. It is a static site stored in `viewer/` inside the project: it works offline and can be copied to another machine together with the folder.

The top bar has the main sections and the search box (**Search objects (/)**, keyboard shortcut `/`), which finds objects by name from any page.

Overview



Project overview

The front page shows where the KB came from (source, input, date, language), the result of the quality check and the overall figures: objects, objects with a document, clusters, links, average confidence and narrative status. Below are the **confidence** distribution, the table by object type, the largest clusters and the links to the KB reports (summary, rebuild roadmap, validation, acceptance).

Confidence (0 to 100) measures how much evidence the engine found to describe each object: `high`, `medium` or `low`. Low confidence does not mean the object is wrong, only that it has little structure to analyse (for example, an empty procedure or a configuration object).

Objects

The screenshot shows the 'Objects' page in UltraGX. At the top, there's a navigation bar with 'UltraGX', 'FacturaDemo', 'Overview', 'Objects', 'Data model', 'Clusters', 'Call trees', 'KB data flow', and 'Narrative'. A search bar on the right says 'Search objects (/)'. Below the navigation bar, there are tabs for 'Type', 'Module', and 'Cluster'. A sidebar on the left lists various object types like 'DataProvider', 'DataSelector', 'Domain', etc. The main area has a filter bar with 'filter...', 'Type', 'Module', 'Folder', 'Own / Library', 'Confidence', 'Narrative', and a count '645 / 645'. Below this is a text box to 'Write a note about this object...' with an 'Add note to selected objects' button. The main table lists objects with columns: Objects, Type, Module, Folder, Cluster, Profile, Confidence, Narrative, and Notes. The table contains 20 rows of object data.

Objects	Type	Module	Folder	Cluster	Profile	Confidence	Narrative	Notes
GetHomeModulesSample	DataProvider	WWPBaseObjects	Library	cluster_014	simple_provider	83	high	pending
GetHomeSampleData	DataProvider	WWPBaseObjects	Library	cluster_002	simple_provider	77	high	pending
GetHomeSampleDataService	DataProvider	WWPBaseObjects	Library	cluster_002	simple_provider	77	high	pending
GetHomeSampleNameValueData	DataProvider	WWPBaseObjects	Library	cluster_002	simple_provider	77	high	pending
GetMainHomeModulesSample	DataProvider	WWPBaseObjects	Library	cluster_014	simple_provider	77	high	pending
GetNotificationSamples	DataProvider	WWPBaseObjects	Library	cluster_002	simple_provider	83	high	pending
GetSomeHomeModulesSample	DataProvider	WWPBaseObjects	Library	cluster_014	simple_provider	77	high	pending
MenuOptionsData	DataProvider	WWPBaseObjects	Library	cluster_002	simple_provider	91	high	pending
SecGAMGetAdvancedSecurityWWPFunctionalities	DataProvider	WWPBaseObjects	Library	cluster_002	simple_provider	77	high	pending
ClienteWWDS	DataSelector	Root Module	Own	cluster_009	parameterized_selector	76	high	pending
FacturaWWDS	DataSelector	Root Module	Own	cluster_007	parameterized_selector	76	high	pending
ProductoWWDS	DataSelector	Root Module	Own	cluster_010	parameterized_selector	76	high	pending
APIAuthorizationStatus	Domain	GeneXus	Library	cluster_169	enumerated_domain	100	high	pending
ActionTexts	Domain	Root Module	Own	cluster_001	enumerated_domain	100	high	pending
Address	Domain	GeneXus	Library	cluster_006	base_domain	92	high	pending
ApplicationState	Domain	GeneXus	Library	cluster_146	enumerated_domain	100	high	pending
AudioAPISessionType	Domain	GeneXus	Library	cluster_173	enumerated_domain	100	high	pending
BarcodeType	Domain	GeneXus	Library	cluster_179	enumerated_domain	100	high	pending
BeaconProximity	Domain	GeneXus	Library	cluster_012	enumerated_domain	100	high	pending
CUIT	Domain	Root Module	Own	cluster_001	validated_domain	92	high	pending
CallTargetSize	Domain	GeneXus	Library	cluster_171	enumerated_domain	100	high	pending

Object list

The list of all objects, with filters by text, **Type** (including the **Business objects** option, which keeps Transactions, Web Panels, Work Panels, Procedures, Reports, Query Objects and SDTs), **Module**, **Folder**, **Own / Library**, **Confidence** and **Narrative** status. Above the list there is a box to add a note to the selected objects (see [User notes](#)).

The **sidebar**, present on every page, groups the objects by **Type**, **Module**, **Location** (the same tree of modules and folders as the GeneXus *KB Explorer*, when the KB has folders) or **Cluster**. In **Types** you choose which types to show and, if the KB has objects from imported modules, **Hide library objects** leaves them out.

Object page

The screenshot shows the 'Object page' for 'Factura' in UltraGX. The top navigation bar is the same as the previous screenshot. The left sidebar is expanded to show 'Factura' under 'Transaction'. The main area has tabs for 'Call tree', 'Data flow', 'Cluster', 'Knowledge pack', 'Data model', 'Source .md', and 'Analysis json'. Below these are two sections: 'Uses (8)' and 'Used by (11)', each with a list of related objects. At the bottom, there's a 'User notes' section with a text box and a 'Save note' button. On the right, there's a 'User notes' sidebar with a list of notes and a search bar.

Uses (8)	Used by (11)
• Cliente data	• ClienteFacturaWC call
• FacturaLoadDVCombo nominal_reference	• ClienteView data
• FacturaWW call	• FacturaFacWC nominal_reference
• LoadWWPContext call	• FacturaGeneral call
• ProductoPrompt nominal_reference	• FacturaLoadDVCombo candidate
• SecGAMFunctionalityKey data	• FacturaPrompt data
• WWPContext data	• FacturaWW call
• WWPTransactionContext data	• FacturaWWExportReport data
	• ListWWPPrograms data
	• ProductoFacFacturaWC call
	• WorkWithPlusFactura parent

Object document

At the top are the object's identity (type, module, folder, cluster, functional profile, data access, confidence), its relationships (**Uses** and **Used by**, with links) and the **User notes**. Then comes the

document, with the table of sections on the right:

Section	What it contains
Summary	table with the key facts of the object
What it does	prose description of its purpose
Data model	(transactions) levels, attributes with their role (key, foreign key, formula, inferred) and entity-relationship diagram
Business rules	each rule, explained one by one
Behavior (pseudocode)	what each event and subroutine does, in readable pseudocode
Implicit SQL (per For each)	the query GeneXus generates for each For each : base table, joins, conditions and order
Parameters and variables	the parm rule and the variables with their type
User interface	fields, grids and their base table, buttons and actions, dynamic behaviour
Business meaning	AI narrative prose, if it was generated
Cross references	related objects
Limits of this analysis	what the engine could not determine with certainty
GeneXus source (verbatim)	the source code exactly as it is in the KB

At the end, collapsible, is the **Deterministic engine annex**: the complete technical evidence of the analysis (findings, risks, score, traceability). It is useful for auditing a conclusion; you do not need it to read the object.

Sections vary by type: a Data Provider shows its output, an SDT its structure, a Domain its values, and so on.

Data model

Transaction data model Factura

- Factura · Module: Root Module · back to the data model

Levels and tables

Level	Table	Keys	Columns	References
Factura	Factura	FacId	13	Ciente
Fac	FacturaFac	FacId, FacSec	5	Factura, Producto

Entity relationship diagram

Cliente

Id_Numeric_12

CiId

PK

FacClId

Factura

Id_Numeric_12

FacId

PK

Factura Numero Interno

Date

FacFecha

Factura Fecha

Numeric_5

FacPrefijo

Factura Prefijo

Character_1

FacLetra

Factura Letra

Numeric_8

FacNumero

Factura Numero

Character_16

FacNoOficial

Factura Numero Oficial

Id_Numeric_12

FacClId

FK

Factura Cliente -> CiId

Nom_VarChar_100

FacCiNom

Factura Razón Social -> CiNom

DateTime

FacUpd

Factura Última Actualización

Nom_VarChar_100

FacUpdUsr

Factura Usuario Actualización

Sts_Character_1

FacSts

Factura Estado

attr

FacUltSec

Factura Última Secuencia

attr

FacTotal

Factura Total

Producto

Id_Numeric_12

ProdId

PK

ProdId

Data model

The entity-relationship diagram of the whole KB, the list of tables and the detail per transaction. Each table has its own page with columns (primary and foreign keys), relationships, the transactions that define it, the objects that use it and a conceptual DDL.

Clusters

A **cluster** is a group of objects that work together (for example, everything related to invoicing). The engine builds them from the dependency graph and the location in the KB. Each cluster has its kind, priority, core object, entry points and members, and three reports: **Knowledge** (purpose, data, risks), **Functional flow** (how work passes from one object to another) and **Rebuild plan**.

The **Rebuild roadmap** (from the Overview) orders the clusters in phases according to their dependencies: first what depends on nothing, then what builds on that.

Call trees and data flow

- **Call trees:** from each entry point, who calls whom. The diagram nodes are clickable.
- **KB data flow:** which objects read and write which tables, for the whole KB, per cluster and per object.

Very large diagrams are not drawn automatically: the **Render diagram** button appears, and you can also see the text with **Diagram source**.

Narrative

AI narrative status per object: `ok`, `degraded`, `error` or pending. See [AI narrative](#).

The output folder

Everything the viewer shows is in files inside the project folder:

Path	Content
<code>README.md</code>	Entry point: how to use the folder, map, glossary, clusters, tables and complete inventory. Always in English
<code>run_summary.json</code>	How it was generated: source, input, language, counts, licence, acceptance
<code>kb_index.json</code>	One row per object: type, module, folder, profile, confidence, cluster and <code>doc_path</code> , the exact path of its document
<code>kb_graph.json</code>	Dependency graph: incoming and outgoing relationships of each object
<code>kb_summary.md</code> / <code>.json</code>	KB metrics
<code>kb_clusters.json</code> , <code>kb_clusters_summary.md</code>	Clusters
<code>global_rebuild_roadmap.md</code> / <code>.json</code>	Rebuild roadmap by phases
<code>acceptance_report.md</code> / <code>.json</code>	Final quality check
<code>objects/</code>	One <code><Type>.<Name>.md</code> and one <code>.json</code> per object, in folders that mirror the KB modules and folders
<code>knowledge/</code>	Knowledge pack of each object
<code>datamodel/</code>	Data model: index, one page per table and per transaction, <code>kb_datamodel.json</code>
<code>clusters/</code>	Knowledge, flow and rebuild plan of each cluster
<code>calltree/</code> , <code>dfd/</code>	Call trees and data flow diagrams
<code>narrative/</code>	AI narrative result, one JSON per object
<code>annotations/</code>	User notes
<code>viewer/</code>	The viewer. It is regenerated: do not edit it by hand
<code>_pipeline/</code> , <code>_source_text/</code>	Internal use: snapshots to detect changes and, with the live source, the text export

The documents are standard Markdown with Mermaid diagrams: they read well on GitHub, GitLab, VS Code or Obsidian.

AI narrative

The pipeline documents the KB with deterministic rules. The **narrative** is an optional step that adds prose written by a language model to each document: the purpose of the object, a summary, the intent of each event, the rules explained in business terms, migration notes and a glossary.

What sets it apart from pasting the code into a chat is **validation**: the model receives the already analysed document (not raw GeneXus) and UltraGX checks that every attribute, table, variable or object it cites really exists in that object. If the model invents a name, it is asked to correct it; if it insists, the prose is not published.

The narrative runs on a project **already generated** by the pipeline.

UltraGX Projects Viewer Run pipeline **Narrative** Jobs Settings

Narrative (LLM prose validated against the facts)

Project
FacturasDemo

Connection
Gemini (company)

Gemini (company) + New Delete

Provider
Google (Gemini)

Model
gemini-3.8-flash

Base URL (openai-compatible)
https://generativelanguage.googleapis.com/v1beta/openai/

API key of this connection (stored in the settings file)
.....

Language
English

Concurrency
4

Types (comma separated, empty = all)
procedure,transaction,webpanel

Limit (0 = all)
0

Document characters sent to the model (lower it to 16 000 for a local model with a small context)
60000

Narrative tab

Connections

A **connection** is a provider, a model and its credentials, saved under a name. You can have several (for example, a company account and a local model) and choose which one to use on each run.

1. In **Connection**, click **+ New** and type a name.
2. Choose the **Provider**. **Model** and **Base URL** are filled in with default values that you can change.
3. Paste the **API key of this connection**, or tick **Runs locally** if you use a model on your computer or your network.

4. Changes are saved when you run. **Delete** removes the selected connection.

Provider	Default model	Notes
Anthropic (Claude)	claude-sonnet-5	Very good prose quality and respect for names
OpenAI (GPT)	gpt-4o-mini	
Google (Gemini)	gemini-3.8-flash	Large context and fast; a good choice for large KBs
DeepSeek, Zhipu (GLM), Moonshot (Kimi), Alibaba (Qwen)	depends on the provider	Support Runs locally if the model is on your own infrastructure
Google (Gemma)	gemma3:latest	Designed to run locally
OpenRouter	a model from its catalogue	A single key for many models
mock (no model)	—	Generates test text at no cost; useful to check the flow

Local models

With **Runs locally**, the API key is hidden and the **Base URL** points to your OpenAI-compatible server: `http://localhost:11434/v1` for Ollama, `http://localhost:1234/v1` for LM Studio, or the address of another machine on your network. Install and download the model on that server before running (for example `ollama pull qwen3.6`).

With a local model **no document leaves your network**. The trade-off is speed and context size: if the model has a small context window, lower **Document characters sent to the model** to 16,000 or less.

Where keys are stored

API keys are stored in the UltraGX settings file (`%APPDATA%\UltraGX\settings.json` on Windows, `~/.config/UltraGX/settings.json` on Linux and macOS), inside your user profile. They are not written to the output folder or sent to UltraGX. Treat that file as a secret.

Run options

Option	What it does
Project	the project to narrate
Language	language of the prose (ideally the same as the project)
Concurrency	objects in parallel. With cloud providers, 4 to 8 speeds things up a lot; with a local model, 1
Types	object types to narrate, separated by commas (for example <code>procedure,transaction,webpanel</code>). Empty = all types with logic
Limit	maximum number of objects in this run (0 = all). Useful to try with 10 objects before launching the whole KB
Document characters sent to the model	maximum size of each document sent (60,000 by default). Longer documents are truncated

With **Types** empty, UltraGX **does not narrate declarations**: attributes, domains, SDTs, subtype groups, external objects, data views and pattern settings. Their deterministic document already says everything there is to say and narrating them only spends tokens. If you still want to narrate any of them, write it in **Types**.

Running, following and resuming

Click **Run / resume**. The log shows each object with its result. Each response is merged into its document **immediately**: you can open the object in the **Viewer** (with **Reload**) while the run continues, and the full viewer is regenerated periodically and at the end.

If you stop the run or it is interrupted, click **Run / resume** again: objects already narrated are not sent again.

Results per object

Status	What it means	Published
ok	the prose passed validation	yes, in the Business meaning section and in the corresponding sections
degraded	after the retries, the model kept citing names that do not exist in the object	no; the document shows a notice with the invented names
error	the provider failed (invalid key, quota exhausted, invalid response, timeout)	no

Retry degraded / errors processes only those objects again. Transient provider errors (rate limits, 5xx errors, network drops) are already retried automatically with progressive back-off during the run.

The status of all objects is on the viewer's **Narrative** page and in the **Narrative** column of the object list.

Costs and privacy

- Each object is one call to the model with its full document (up to the character limit). To estimate the cost, run with **Limit** 20, look at the tokens the log reports at the end and extrapolate to the number of objects with logic in your KB.
- With cloud providers, each object's document (including its GeneXus code) is sent to that provider with your account and under its terms. If the KB has confidentiality restrictions, use a local model or a provider approved by your organisation.
- In the demo edition only the sample objects are narrated.

User notes

Notes record what the analysis cannot know: that a procedure is no longer used, that a rule follows a regulation, who the functional contact for a module is or what the team decided for the migration. They live inside the documentation, so an AI agent reads them too.

Notes are added **from the application (Viewer tab)**. If you open the viewer in a separate browser, they are visible but cannot be created or deleted.

Note on an object



Notes on an object page

1. Open the object in the **Viewer**.
2. In **User notes**, write the note and, if you like, your name in **Author**.
3. Click **Save note**.

The note appears on the page straight away, with date and author. **Delete** removes it.

Note on several objects at once

1. In **Objects**, filter the list (for example, by folder or by cluster).
2. Tick the boxes in the last column, or the header box to **Select visible**.
3. In the bar that appears above, write the note and click **Add note to selected objects**.

Each object receives its own copy of the note. The **Notes** column of the list marks which objects have notes.

Where they are stored

- In `annotations/` inside the project, one JSON file per object. It is the source of truth: copy that folder if you want to take the notes to another machine.
- In the object's Markdown document, **User notes** section.
- In the object's JSON, `user_notes` field.

When you run the pipeline again on the same project (for example, with a newer export of the KB), the notes are merged into the regenerated documents again. If an object has disappeared from the KB, its notes file stays in `annotations/` but is no longer shown.

Migrating with an AI agent

The UltraGX output is written so that a coding agent (Claude Code, Codex, Cursor, Antigravity or another) can rebuild the application in another technology without opening GeneXus. The agent does not have to interpret GeneXus code from scratch: it reads rules that are already explained, pseudocode per event, the implicit SQL of each `For each`, the data model with its keys and a rebuild order.

This chapter describes a way of working that works well. It is not the only one, but it avoids the most common mistakes: asking the agent for everything at once, or letting it invent structure that the KB already defines.

1. Preparing the workspace

1. Generate the documentation with the **full edition** and, if you can, with the narrative at `ok` for the business objects. Use the language your team works in.
2. Create the repository of the new application **separate** from the UltraGX output.
3. Make the output available to the agent as read-only material: for example, a sibling folder (`../kb-docs/`) or a submodule. There is no need to copy `viewer/`, `_pipeline/` or `_source_text/`.
4. In the new repository, write the agent's instructions file (`CLAUDE.md` , `AGENTS.md` or the Cursor rules) with:
 - the target technology and its conventions (language, framework, database, folder structure);
 - the location of the KB documentation and the instruction to read its `README.md` first;
 - the rule that table, attribute and rule names come from the documentation and are never invented;
 - that **user notes** take precedence over inferred statements.

Example block for that file (with documents in English, the sections are called **User notes** and **Limits of this analysis**, as below; adjust the names if your documents are in another language):

```
## Legacy system knowledge
```

```
The GeneXus application being migrated is documented in ../kb-docs (UltraGX output, read-only).
```

```
Start with ../kb-docs/README.md. Locate objects through kb_index.json (doc_path column).
```

```
Table and column names, keys and business rules come from those documents; never invent them.
```

```
"User notes" sections override inferred statements. When a document lists something under
```

```
"Limits of this analysis", ask before assuming.
```

2. Reading order

The output's `README.md` tells the agent how to go through it. In short:

1. `README.md` , `kb_summary.md` and `kb_clusters_summary.md` : what there is and how it is grouped.

2. `datamodel/index.md`: tables, keys and which transaction defines each table.
3. `global_rebuild_roadmap.md`: in what order to rebuild.
4. For each cluster: `clusters/cluster_NNN.knowledge.md`, `.flow.md` and `.rebuild.md`.
5. For each object: its document in `objects/` (path in `kb_index.json` → `doc_path`).
6. To follow relationships: `kb_graph.json` or the **Cross references** section of each document.

3. Working by phase and by cluster

1. **Data model first.** Ask the agent for the database schema and the entities based on `datamodel/`. Check it against the real database before going on: everything else builds on this.
2. **Follow the roadmap.** Take the first phase of `global_rebuild_roadmap.md` and, within it, one cluster at a time.
3. **One cluster, one task.** For each cluster, the agent reads its rebuild plan and the documents of its objects, proposes the design, implements it and writes tests based on the business rules.
4. **Close the cluster** before moving to the next one: tests green, rules reviewed one by one and user notes with the decisions taken (they remain for the agent's next sessions).

4. Sample prompts

Orientation

Read `../kb-docs/README.md`, `kb_summary.md` and `kb_clusters_summary.md`. Summarise what the application does, its functional areas and the 10 most important tables. Do not write any code yet.

Data model

Based on `../kb-docs/datamodel/`, generate the database migrations for PostgreSQL and the ORM entities. Keep the names, types, primary keys and foreign keys. List any table whose definition is incomplete.

One cluster

Implement cluster `cluster_004` according to `clusters/cluster_004.rebuild.md`. Read the document of each member object and reproduce its business rules and the implicit SQL of its For each commands. For the screens, use the user interface section. Write one test per business rule and flag anything you cannot reproduce with certainty.

A single object

Look up the procedure `CalcularDescuento` in `kb_index.json`, read its document and reimplement it as a service. Explain each branch of the pseudocode in a comment in the corresponding test.

Verification

Compare the InvoiceService service with objects/Facturacion/Transaction.Factura.md: list every business rule in the document and state where it is implemented or whether it is missing.

5. Good practices

- **Limited context.** Do not ask the agent to read the whole folder: with `kb_index.json` and `doc_path` it finds what it needs for each task.
- **Confidence.** Prioritise human review of objects with a low `confidence_level` and of whatever appears under **Limits of this analysis**.
- **Library and pattern objects.** Objects marked as library (modules imported from GeneXus or third parties) and pattern scaffolding (Work With, K2BTools) are not migrated as they are: they are replaced by the equivalents of the target framework.
- **The KB is still alive.** If the GeneXus application receives changes during the migration, export again and run the pipeline on the same project; notes are kept and the differences are visible in version control if the output is versioned.

Command line

Everything the application does is carried out by the engine, `ultragx-engine`, which can also be used directly from a terminal: to automate the documentation of several KBs, schedule it on a server or integrate it into a continuous integration process. It uses the same licence as the application.

Where the engine is

System	Path
Windows (installer)	<code>%LOCALAPPDATA%\Programs\UltraGX\ultragx-engine.exe</code>
Linux (deb / rpm)	<code>/opt/ultragx/ultragx-engine</code>
Linux (AppImage)	<code>./UltraGX-<version>-x86_64.AppImage engine <command> ...</code>
macOS	<code>/Applications/UltraGX.app/Contents/MacOS/ultragx-engine</code>

The examples use `ultragx-engine`; replace it with the corresponding path or add the folder to the `PATH`.

Commands

version and machine-id

```
ultragx-engine version      # engine version
ultragx-engine machine-id  # machine id, to request a licence bound to it
```

license

```
ultragx-engine license      # licence status, as JSON
ultragx-engine license activate --code ULGX-XXXX-XXXX  # activates a licence with a code
```

The demo can also be requested without the application, in two steps:

```
ultragx-engine license register --first-name Ana --last-name Pérez --company "My Company" \
    --email ana.perez@mycompany.com --gx-version 18 --kb-size 3000 --lang en
ultragx-engine license confirm --email ana.perez@mycompany.com --code 123456
```

pipeline

Documents a KB from an export.

```
ultragx-engine pipeline --source xpz --input C:/exports/MyKB.xpz --out D:/ultragx/MyKB --lang
en
ultragx-engine pipeline --source gxtxt --input D:/exports/MyKB_text --out D:/ultragx/MyKB --
lang en --no-ref
```

Option	Description
<code>--source</code>	<code>xpz</code> (<code>.xpz</code> or <code>.xml</code> file) or <code>gxttext</code> (folder with <code>src/</code> and <code>ref/</code>)
<code>--input</code>	input file or folder
<code>--out</code>	project folder (created if it does not exist)
<code>--lang</code>	language of the documents: <code>en</code> , <code>es</code> or <code>pt-BR</code>
<code>--no-ref</code>	with <code>gxttext</code> , skips the objects of referenced modules
<code>--no-viewer</code>	does not generate the viewer (faster; it can be generated later)

Progress is written to standard error and, when it finishes, the run summary is printed as JSON on standard output.

export

Opens a KB with the installed GeneXus (Windows only, with the connector), exports it to text and runs the pipeline.

```
ultragx-engine export --kb-path "C:/Models/MyKB" --kb MyKB --out D:/ultragx/MyKB --lang en
```

Option	Description
<code>--kb-path</code>	KB folder (the one that contains the <code>.gxw</code>)
<code>--kb</code>	short name of the KB
<code>--out</code>	project folder
<code>--gx-version</code>	forces a GeneXus installation (<code>18</code> , <code>17</code> , <code>XEv3</code> ...); by default it is deduced from the KB
<code>--names</code>	exports only some objects: names or selectors such as <code>type:Transaction</code>
<code>--no-reference</code>	skips referenced modules
<code>--skip-pipeline</code>	exports only, without documenting
<code>--timeout</code>	maximum export time in seconds (3600 by default)
<code>--lang</code> , <code>--no-viewer</code>	same as in <code>pipeline</code>

viewer

Regenerates a project's viewer without analysing the KB again.

```
ultragx-engine viewer --out D:/ultragx/MyKB --lang en
```

narrative

Runs the AI narrative on an already documented project. On the command line, credentials are taken from environment variables: `ANTHROPIC_API_KEY` for Anthropic, `OPENAI_API_KEY` (and `OPENAI_BASE_URL` for another server) for the OpenAI-compatible providers, including Gemini, OpenRouter and local models.

```
export OPENAI_API_KEY=...          # in PowerShell: $env:OPENAI_API_KEY = "..."
ultragx-engine narrative run --out D:/ultragx/MyKB --provider openai \
    --base-url https://generativelanguage.googleapis.com/v1beta/openai/ --model gemini-3.8-
flash \
    --lang en --concurrency 8
```

Option	Description
<code>--provider</code>	<code>anthropic</code> , <code>openai</code> (any compatible server) or <code>mock</code>
<code>--model</code> , <code>--base-url</code>	model and server URL
<code>--lang</code>	language of the prose
<code>--concurrency</code>	objects in parallel (4 by default)
<code>--types</code>	only these types, separated by commas
<code>--all-types</code>	also includes declarations (attributes, domains, SDTs...)
<code>--only</code>	only these objects, as <code><Type>.<Name></code> separated by commas
<code>--limit</code>	maximum number of objects
<code>--no-resume</code>	processes objects already narrated again
<code>--max-doc-chars</code>	document characters sent (60,000 by default; 16,000 with a local server if not specified)
<code>--no-source</code>	does not send the original GeneXus code to the model
<code>--timeout</code>	seconds to wait for each response (300 by default)
<code>--viewer-every</code>	seconds between viewer regenerations during the run (600 by default; 0 = only at the end)

notes

Adds or deletes user notes without the application.

```
ultragx-engine notes add    --out D:/ultragx/MyKB --json note.json
ultragx-engine notes delete --out D:/ultragx/MyKB --stem Procedure.CalcularDescuento --id
n171234567890000
ultragx-engine notes apply  --out D:/ultragx/MyKB
```

`note.json` has the form `{"stems": ["Procedure.CalcularDescuento"], "text": "...", "author": "Ana"}`. `apply` merges all the notes into the documents again.

Exit codes

Code	Meaning
0	finished successfully
1	the narrative finished with objects in <code>error</code>
2	incorrect usage (invalid command or options) or, in <code>export</code> , the KB could not be opened

Code	Meaning
3	licence missing, invalid, expired or revoked

Example: documenting every KB in a folder

```
$engine = "$env:LOCALAPPDATA\Programs\UltraGX\ultragx-engine.exe"
Get-ChildItem D:\exports\*.xpz | ForEach-Object {
    & $engine pipeline --source xpz --input $_.FullName --out "D:\ultragx\${$_}.BaseName)" --
lang en
    if ($LASTEXITCODE -ne 0) { Write-Warning "${$_}.Name) finished with code $LASTEXITCODE" }
}
```

Troubleshooting and FAQ

Installation and start-up

Windows shows "Windows protected your PC". It is SmartScreen with an installer that still has little reputation. Click **More info → Run anyway**. If your organisation blocks that option, ask your IT support to allow the installer.

The UltraGX window opens blank or does not open (Windows). The Microsoft Edge WebView2 runtime is missing or damaged. Install it from the Microsoft page ("WebView2 Runtime", *Evergreen* installer) and open UltraGX again. The UltraGX installer adds it automatically.

The application does not open on Linux. Run it from a terminal (`ultragx` or the AppImage) to see the error. If it mentions `libwebkit2gtk`, install `libwebkit2gtk-4.1-0` (Debian/Ubuntu) or `webkit2gtk4.1` (Fedora).

macOS says UltraGX "cannot be opened". The macOS build is not signed with an Apple Developer ID. Double-click **Allow UltraGX.command** in the disk image after dragging UltraGX to Applications (or run `xattr -cr /Applications/UltraGX.app` in Terminal), or allow it in **System Settings → Privacy & Security → Open Anyway**.

Licence

"The licence service could not be reached" appears. The computer cannot reach `https://licenses.ultra-gx.com`. Check the internet connection; on corporate networks, ask for HTTPS to that domain to be allowed (also through the proxy). The other registration messages are explained in [First start and licence](#).

The run ends with "A valid license file (ultragx.lic) is required..." or code 3. The licence is missing, has expired or has been revoked. Check **Settings → License**. If the demo has ended, write to us to move to the full edition.

In the demo, no object came out detailed. The run could not check with the licence service and continued without a sample. The reason is in the notice in the `README.md` and in the viewer. Fix the connection and run the pipeline again.

I changed computers. The licence is bound to the machine where it was activated. Write to us with the new **This machine id** to move it.

Documenting a KB

The "live" source appears disabled. **Settings → Environment** shows why: *GeneXus connector: not found* (Linux, macOS or a copy without the connector) or *GeneXus installed: none detected*. In those cases export the KB to XPZ on the machine where GeneXus is.

The live export does not progress or fails when opening the KB. The KB is open in the GeneXus IDE or in another tool: close it and run again. If the log repeats the same object for several minutes,

the connector is still extracting a large object; on KBs with thousands of objects the export can take more than an hour.

I chose the wrong GeneXus version. Leave **auto (from the KB)**. Force an installation only if the KB opens with a different version from the one detected.

Some SDTs or transactions appear without structure. If the KB was documented with a version earlier than 0.2.5, run it again with the current version (with the live source, the export too).

Acceptance ended in warning. The documentation was generated in full; the quality check found something to review. The detail is under **Acceptance** in the viewer. **failed** indicates a problem worth reporting.

The summary warns library_flag_suspect. Almost all objects were flagged as library, which indicates an export that did not separate own objects from referenced ones properly. Export the KB again (with the current connector, if it is live).

Paths or folder names are too long (Windows). The output mirrors the modules and folders of the KB. Use a short output folder, such as `D:\ultragx`.

The output folder is in OneDrive / Dropbox and everything is slow. Synchronisation processes thousands of files at once. Use a folder outside synchronised folders.

Narrative

Message in the log	Cause	Solution
401 , 403 , invalid API key	the connection key is not valid or has no permission for that model	check the connection's API key
404 , model not found	the model name does not exist at that provider	correct Model with a name from the provider's catalogue
429 , quota, rate limit	provider usage limit	the engine retries on its own; if it persists, lower Concurrency or check your account quota
connection refused with a local model	the local server is not running or the URL is wrong	start Ollama / LM Studio and check the Base URL
many degraded	the model cites names that do not exist	try a more capable model; with local models, increase the context size

The prose does not appear in the viewer. Click **Reload**. Only objects in **ok** are published.

Where to look

What	Where
Log of a run	Jobs → Log while the application is open
Result of the run	<code>run_summary.json</code> and <code>acceptance_report.md</code> in the project folder
Narrative status	<code>narrative/checkpoint.json</code> and one JSON per object in <code>narrative/</code>
Settings and licence	<code>%APPDATA%\UltraGX</code> (Windows) or <code>~/ .config/UltraGX</code> (Linux and macOS)

Support

Write to us at contact@ultra-gx.com with:

- the UltraGX version (**Settings → Environment → App**) and the operating system;
- the error message or the end of the job log;
- the project's `run_summary.json` and `acceptance_report.md`, if the problem is with a run.

Do not send the XPZ, the KB code or the complete output folder unless we ask for it and your organisation authorises it.

Frequently asked questions

Does UltraGX modify my KB? No. It only reads: the connector has no write path at all and with an XPZ it does not even touch the KB.

Do I need GeneXus installed? Only for the live source. With an XPZ or a text export, no.

Which GeneXus versions does it support? From 7.5 to 18 via XPZ/XML; from X Evolution 1 to 18 live.

Can I document the same KB many times? Yes. With the same project name it is regenerated in the same folder and notes and narrative are kept. In the demo, the 25-object sample is always the same.

Can I share the documentation? Yes: the project folder is self-contained and the viewer works offline. Bear in mind that it contains the KB code.

Is the source code sent anywhere? Only if you use the narrative with a cloud provider, and only to that provider. See [AI narrative](#).

Glossary

Term	Meaning
Acceptance	Quality check the engine runs at the end of each pipeline: <code>passed</code> , <code>warning</code> or <code>failed</code> .
Business object	Transactions, Web Panels, Work Panels, Procedures, Reports, Query Objects and SDTs: where the application logic lives.
Call tree	Diagram of who calls whom, starting from an object or an entry point.
Cluster	Group of objects that work together in the same functional area, inferred from the dependency graph and the location in the KB.
Confidence	Score from 0 to 100 (<code>high</code> , <code>medium</code> , <code>low</code>) indicating how much evidence backed the analysis of an object.
Connection	AI provider, model and credentials saved under a name, for the narrative.
Connector	UltraGX component for Windows that opens a KB with the installed GeneXus, read-only, and exports it to text.
Data flow diagram (DFD)	Diagram of which objects read and write which tables.
Declaration	Object that only defines a type, a structure or a setting (attribute, domain, SDT, external object, pattern settings). Not narrated by default.
<code>degraded</code>	Narrative result for an object when the model cited names that do not exist in it. Not published.
Demo / full edition	Licence types. The demo lasts 10 days and details 25 objects; the full edition documents everything offline.
Deterministic annex	Last part of each object document, with the complete technical evidence of the analysis.
Engine	<code>ultragx-engine</code> , the program that analyses the KB and writes the documentation. The application runs it for you.
Functional profile	Classification of what an object does (for example, data maintenance, query, batch process).
Implicit SQL	The query GeneXus generates for a <code>For each</code> , reconstructed by the engine: base table, joins, conditions and order.
KB (Knowledge Base)	The GeneXus knowledge base: all the objects of an application.
Knowledge pack	Structured summary of an object: functional understanding, interaction model, data usage, dependencies, risks and gaps.
Library object	Object from an imported module (from GeneXus or third parties), not developed in the KB.
Live (source)	Reading the KB directly with the installed GeneXus, through the connector.
Narrative	Prose written by a language model and validated against the facts of the object.
Pipeline	The complete run of the engine on a KB: reading, analysis, relationships and writing.
Project	The output folder of a documented KB.

Term	Meaning
Pseudocode	Readable translation of each event and subroutine of the object.
Rebuild roadmap	Suggested order for rebuilding the application, in phases, according to the dependencies between clusters.
Stub	In the demo, summary document of an object outside the sample.
Text export	GeneXus 17 and 18 format that stores each object as a text file, in <code>src/</code> and <code>ref/</code> folders.
User notes	Comments the team adds to objects; they are kept between runs and take precedence over inferred statements.
Viewer	Static website generated for each project, browsable offline.
XPZ	GeneXus object export file (compressed XML).