Zebra
Aurora™ Vision

Aurora Vision Studio 5.7

Appendices

Created: 11/10/2026

Product version: 5.7.5.80527

Table of content:

- Backward Compatibility Policy
- Quick Start Guide for the Users of LabVIEW
- Quick Start Guide for the C/C++ Programmers
- Deep Learning – Getting Started

Backward Compatibility Policy

Programs created in Aurora Vision Studio are fully backward compatible within the same release number.

Between consecutive minor revisions (3.1, 3.2, ...) the following changes might be anticipated:

- User Filters might have to be rebuilt.
- Data files (e.g. Template Matching models) might have to be recreated.
- Generated C++ code might have to be re-generated.

Quick Start Guide for the Users of LabVIEW

Both Aurora Vision Studio and LabVIEW are based on the idea that the data-flow programming model can be very useful in data-processing applications. This makes the two applications similar on the general level, but still they are very different in the details. Aurora Vision Studio has been primarily designed for machine vision engineers, whereas LabVIEW is advertised as "software ideal for any measurement or control systems". This results in a completely different design of the user interface with Aurora Vision Studio concentrating on more sequential algorithms and Data Previews accessible directly within the Main Window.

If you have used LabVIEW before, below you will find a list of key differences that will allow you to start working with Aurora Vision Studio more quickly by mapping the new concepts to what you already know:

LabVIEW	Aurora Vision Studio	Notes
Nodes	Filter Instances	In both environments these elements have several inputs and outputs, and are the basic data processing elements.
Wires	Connections	Connections in Aurora Vision Studio encompass more program complexity than wires in LabVIEW. Like in LabVIEW there are basic connections and connections with data conversion (LabVIEW: coercion). There are, however, also array connections that transmit data in a loop and conditional connections that can make the target filter not executed at all.
Basic Data Types		Aurora Vision Studio has two numeric data types: Integer and Real, both 32-bit. There are also Booleans (LabVIEW: *Boolean*), Strings, File (LabVIEW: *Path*) and enumerated types.
Arrays and Clusters	Arrays and Structures	Arrays and Structures are more similar to the corresponding elements of the C language. Arrays in Aurora Vision Studio can be multi-dimensional, e.g. one can have arrays of arrays of arrays of integer numbers. Structure types are predefined and their elements can be "bundled" and "unbundled" with appropriate Make and Access filters.
Local Variables	Labels	The programming model of Aurora Vision Studio enforces the use of data flow connections instead of procedural-style variables. Instead of local variables you can use labels, that replace connections visually, but not actually.
Global Variables	Global Parameters	Global Variables in LabVIEW are recommended mainly for passing information between VIs that run simultaneously. In Aurora Vision Studio this is similar – global parameters should be used to communicate between Parallel Tasks and with HMI Events.
Dynamic values / Polymorphic VIs	Generic Filters	Generic Filters of Aurora Vision Studio are more similar to templates of the C++ programming language. The user specifies the actual type explicitly and thus the environment is able to control the types of connections in a more precise way.
Waveform	Profile	A sequence of numeric values that can be depicted with a 2D chart is called Profile.
Virtual Instrument (VI, SubVI)	Macrofilter	A macrofilter is a sequence of other filters hidden beyond an interfaces of several inputs and outputs. It can be used in many places of a program as it was a regular filter. Macrofilters do not have their individual front panels. Instead, the environment of Aurora Vision Studio is designed to allow output data preview and input data control.
Front Panel	HMI	In Aurora Vision Studio, HMI (Human-Machine Interface) is created for the end user of the machine vision system. There is thus single HMI for a project. There are no blocks in the Program Editor that correspond to HMI controls. The connections between the algorithm and the HMI controls are represented with "HMI" labels.

For Loop, While Loop	Array Connections, Task Macrofilter	There are two methods to create loops. The first one is straightforward – when the user connects an output that contains an array to an input that accepts a single element, then an array connection is used. A for-each loop is here created implicitly. The second is more like the structures of LabVIEW, but also more implicit – the entire Task macrofilter works in a loop. Thus, when you need a nested loop you can simply create a new Task macrofilter. These loops are controlled by the filters that are used – more iterations are performed when there are filters signaling ability to generate new data.
Shift Registers	Registers	Registers in Aurora Vision Studio are very similar to Shift Registers. One difference is that the types and initial values of registers have to be set explicitly. Step macrofilters preserve the state of registers between subsequent executions within a single execution of the Task that contains them. There are no Stacked Shift Registers, but you can use the LastTwoObjects / AccumulateElements filters instead.
Case Structures	Variant Macrofilter	While Task Macrofilters can be considered an equivalent of the While Loops of LabVIEW, Variant Macrofilters can be considered an equivalent of the Case Structures. Selector Terminals and Cases are called Forking Ports and Variants respectively.
Sequence Structures	–	All macrofilters in Aurora Vision Studio are executed sequentially, so explicit Sequence Structures are not needed.
Controls Palette	HMI Controls	
Functions Palette	Toolbox	
Formula / Expression Nodes	Formula Blocks	Formula Blocks are used to define values with standard textual expressions. Several inputs and outputs are possible, but loops and other C-like statements are not. This feature is thus something between LabVIEW's Expression and Formula Nodes. If you need C-like statements, just use C++ User Filters that are well integrated with Aurora Vision Studio.
Breakpoints	Iterate Current Macrofilter	As there are no explicit loops other than the loops of Task macrofilters, a macrofilter is actually the most appropriate unit of program debugging. One can use the Iterate Current Macrofilter command to continue the program to the end of an iteration of the selected macrofilter.
Error Handling		In Aurora Vision Studio there are no *error in/out* ports. Instead, errors are handled in separate subprograms called Error Handlers. In some cases when an output of a tool cannot be computed, conditional outputs are used. The special value *Nil* is then signaling a special case. This is for example when you try to find the intersection of two line segments which actually do not intersect.
Call Library Function Node	User Filter	User Filters can be used to execute pieces of code written in Microsoft Visual C++. The process of creating and using User Filters is highly automated.

Quick Start Guide for the C/C++ Programmers

Aurora Vision Studio has been created by developers, who were previously creating machine vision applications in C++. We created this product to make this work much more efficient, whereas another our goal was to retain as much of the capabilities and flexibility as possible. We did not, however, simply create a graphical interface for a low level C++ library. We applied a completely different programming paradigm – the Data Flow model – to find the optimum balance between capabilities and development efficiency. Programming in Aurora Vision Studio is more like designing an electrical circuit – there are no statements and no variables in the same way as they are not present on a PCB board. The most important thing to keep in mind is thus that there is no direct transition from C++ to Aurora Vision Studio. You need to **stop thinking in C++ and start thinking in data flow** to work effectively. Automatic C++ code generation is still possible from the data flow side, but this should be considered a one-way transformation. At the level of a data flow program there are no statements, no *if*s, not *for*s.

So, how should you approach constructing a program, when you are accustomed to such programming constructs as loops, conditions and variables? First of all, you need to look at the task at hand from a higher level perspective. There is usually only a single, simple loop in machine vision applications – from image acquisition to setting digital outputs with the inspection results. It is highly recommended to avoid nested loops and use Array Connections instead, which are data-flow counterparts of *for-each* loops from the low level programming languages. For conditions, there is no *if-then-else* construct anymore. There are Conditional Connections instead (data may flow or not), or – for more complex tasks – Variant Macrofilters. The former can be used to skip a part of a program when some data is not available, the latter allow you to create subprograms that have several alternative paths of execution. Finally, there are no variables, but data is transmitted through (usually unnamed) connections. Moreover, Global Parameters can be used to create named values that need to be used in many different places of a program, and Macrofilter Registers can be applied to program complex behaviors and store information between consecutive iterations of the program loop.

Please note, that even if you are an experienced C++ programmer, your work on machine vision projects will get a huge boost when you switch to Aurora Vision Studio. This is because C++ is designed to be the best general purpose language for crafting complex programs with complicated control flow logic. Aurora Vision Studio on the other hand is designed for one specific field and focuses on what is mostly important for machine vision engineers – the ability to experiment quickly with various combinations of tools and parameters, and to visualize the results instantly, alone or in combination with other data.

Here is a summary:

C++	Aurora Vision Studio	Notes
Conditions (the *if* statement)	Conditional Connections, Variant Macrofilters	Aurora Vision Studio is NOT based on the control flow paradigm. Instead, data flow constructs can be used to obtain very similar behavior. Conditional connections can be used to skip some part of a program when no data is available. Variant Macrofilters are subprograms that can have several alternative paths of execution. See also: Sorting, Classifying and Choosing Objects
Loops (the *for* and *while* statements)	Array Connections, Task Macrofilters	Aurora Vision Studio is NOT based on the control flow paradigm. Instead, data flow constructs can be used to obtain very similar behavior. Array connections correspond to *for-each* style loops, whereas Task Macrofilters can be used to create complex programs with arbitrary nested loops.
Variables	Connections, Global Parameters, Macrofilter Registers	Data flow programming assumes no side effects. Computed data is stored on the filter outputs and transmitted between filters through connections. Global Parameters can be used to define a named value that can be used in many different places of a program, whereas Macrofilter Registers allow to store information between consecutive iterations.
Collections (arrays, std::vector etc.)	Arrays	The Array type is very similar to the std::vector<T> type from C++. This is the only collection type in Aurora Vision Studio.

Templates	Generic Filters	As there can be arrays of many different types (e.g. RegionArray, IntegerArrayArray), we also need to have filters that transform arrays of different types. In C++ we have template metaprogramming and the STL library that is based on it. In Aurora Vision Studio we have generic filters, which are very similar to simplified templates from C++ – the type parameter has to be defined when adding such filter to the program.
Functions, methods	Macrofilters	Macrofilters are subprograms, very similar to functions from C++. One notable difference is that macrofilters cannot be recursive. We believe that this makes programs easier to understand and analyze.
GUI Libraries (MFC, Qt, WxWidgets etc.)	HMI Designer	If more complex GUI is needed, the algorithms created in Aurora Vision Studio can be integrated with a GUI written in C++ through C++ Code Generator. See also: Handling HMI Events.
Static, dynamic libraries	Modules	Bigger projects require better organization. As you can create libraries in C++ which can be used in many different programs, you can also create modules (a.k.a. libraries of macrofilters) in Aurora Vision Studio.
Breakpoints	The "Iterate Current Macrofilter" command (Ctrl+F10).	As there are no side effects within macrofilters, there is no need to set breakpoints in arbitrary places. You can, however, run the program to the end of a selected macrofilter – just open this macrofilter in the Program Editor and use the "Iterate Current Macrofilter" command. The program will pause when it reaches the end of the selected macrofilter instance.
Threads		There are no threads in Aurora Vision Studio. Instead, the filters utilize as many processors as possible internally and the HMI (end user interface) is automatically run in parallel and synchronized with the program loop.
Exceptions	Domain Errors, Conditional Outputs	Domain Errors signal unexpected conditions encountered during program execution. They cause the program to stop, so it is crucial to make sure they are not possible. Nevertheless, during development they provide important information about what has to be fixed. If an unexpected condition cannot be easily predicted with careful program construction, then conditional outputs are used and the *Nil* value is returned to signal failure of execution.

Interoperability with C++

Having said, that you can solve even the most challenging machine vision tasks with the data-flow programming model, in real life you most often need to integrate the machine vision solution with a bigger system. This integration most often requires C++ or .NET programming. Aurora Vision Studio comes with several features that make it possible:

- User Filters allow to add your own C++ code in the form of filters of Aurora Vision Studio.
- C++ Code Generator allows to switch from the graphical environment of Aurora Vision Studio to a C++ program based on the AVL.DLL library.
- .NET Macrofilter Interface Generator produces a .NET assembly (a .dll file) with methods corresponding to macrofilters of a program, thus providing a bridge between Aurora Vision Studio and .NET technology

Other remarks:

- The program files of Aurora Vision Studio are based on textual formats. You can use your version control system to store them and monitor their history.

FAQ

Question:

How to mark the end of a loop started with filters such as EnumerateIntegers or Loop?

Answer:

This is by design different than in C++. The loop goes through the entire Task macrofilter, which is a logical part of a program. If you really need a nested loop (which is rare in typical machine vision projects), then create a Task macrofilter for the entire body of the loop. First of all, however, consider array connections. They allow for example to inspect many objects detected on a single image without creating an explicit loop.

Question:

Could you add a simple "if" filter, that takes a boolean value and performs the next filter only if the condition is met?

Answer:

This would be a typical construct in the control-flow based programming languages. We do not want to mix different paradigms, because we must keep our software from becoming too complicated. You can achieve the same thing by using the MakeConditional filter and passing data to the next filter conditionally then. If there is no appropriate data that could be used in that way, then a variant macrofilter might be another solution.

Question:

How to create a variable?

Answer:

There are no variables in data-flow. This is for the same reason you do not see variables on PCB boards or when you look at a production line in a factory. There is a flow instead and connections transmit data (or objects) from one processing element to another. If you need to store information between consecutive iterations, however, then also stateful filters (e.g. AddIntegers_OfLoop, macrofilter registers or appropriate functions in formula blocks can be used.

Deep Learning - Getting Started

Note: The following article concerns the functionalities related to another product: **Deep Learning Add-on**. More information are available here.

Contents

1. Installation guide
2. Aurora Deep Learning Library and Filters
3. Aurora Deep Learning Service
4. Aurora Deep Learning Examples
5. Aurora Deep Learning Standalone Editor
6. Logging
7. References

1. Installation guide

To use Deep Learning Filters, Library or Service with Aurora Vision Studio or Aurora Vision Library, a corresponding version of Aurora Deep Learning must be installed (the best idea is to use the newest versions of both from our website). Before installation, please check your hardware configuration.

Deep Learning is available in two versions:

- GPU version (recommended) - version working with CUDA GPU acceleration. Much faster than CPU counterpart.
- CPU version - uses only CPU, GPU is not required and used. Relatively slow, especially during training phase.

Requirements

- Graphics card compatible with CUDA toolkit. List of compatible devices can be found on this website (all CUDA devices with "Compute Capability" greater than or equal 7.5 and less than or equal 12.0). A minimum of 8 GB of graphic memory is recommended. Display Driver with at least 576.80 version is required (recommended latest display driver version).
- At least 15 GB disk space for program files, SSD recommended.
- At least 16 GB RAM memory.
- 64-bit processor, Intel i5, i7 or better are recommended. AVX support is required.
- Windows 10 or 11.

Known issues

If you are getting **Access Denied** errors during updates (or uninstallation), close all processes that may use previously installed Deep Learning files, such as programs that need the Deep Learning Library, Aurora Vision Studio, Aurora Vision Executor and so on.

2. Aurora Deep Learning Library and Filters

Aurora Deep Learning provides Filters for use in Aurora Vision Studio in the 64-bit version. The 32-bit version is not supported.



Aurora Deep Learning also provides Library in the 64-bit version (the 32-bit version is not supported). It is installed in the *Library* subdirectory of the main installation directory. It contains the following subfolders:

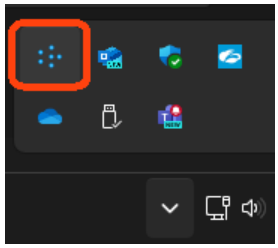
- *bin*, *bin/x64* - a directory containing DLL file (*AVLDL.dll*) for 64-bit applications written in C++. This library is compatible with all supported versions of Microsoft Visual Studio and with Debug|Release configurations. The directory also contains NuGet packages for Aurora Vision Deep Learning Library .NET.
- *include* - a directory containing all header (*.h*) files for Library (*AVLDL.h*)and Training Api (*Api.h*).

- *lib/x64* - a directory containing import library (*AVLDL.lib*) for 64-bit applications. This file needs to be linked into a program that uses Library or Training Api as it serves as an intermediary to *AVLDL.dll*.

The Installer sets the environment variable named AVLDL_PATH5_7 containing the path to the *Library* subdirectory.

3. Aurora Deep Learning Service

The Service is installed into the *Service* subdirectory within the main installation directory. Upon launching the Service, a new icon should appear in the system tray.



Running filters does not require the Service to be running in the background (in fact, this is even discouraged). On the other hand, training a model always requires the service to be running, regardless of the tool used.

Please note: to open the **Deep Learning Editor**, place a relevant Deep Learning filter (DL_DetectFeatures, DL_ClassifyObject, DL_LocatePoints, DL_LocateObjects or DL_DetectAnomalies2) in the Program Editor, enter its Properties and click on the icon next to the **inModelId.ModelDirectory** parameter.

Alternatively, to open the **Deep Learning Editor** from a DL_*_Deploy filter (DL_DetectFeatures_Deploy, DL_ClassifyObject_Deploy, DL_LocatePoints_Deploy, DL_LocateObjects_Deploy or DL_DetectAnomalies2_Deploy), enter its properties and click on the icon next to the **inModelDirectory** parameter.

4. Deep Learning Examples

Examples are installed only if the corresponding component has been selected during installation.

Examples for Aurora Vision Studio are installed into *%ProgramData%/Aurora Vision/Aurora Deep Learning 5.7/Examples* directory. The Deep Learning Add-on installer contains only a few examples. More of them are available with a separate Deep Learning Examples installer available on the website. All Deep Learning Examples are prefixed with "Deep Learning".

Examples for Aurora Vision Library are located in the *Public Documents* system folder (e.g. *C:\Users\Public\Documents\Aurora Deep Learning 5.7\Examples* on Windows 10). Shortcut to this directory can be found in the *Start Menu* after installation.

5.Aurora Deep Learning Standalone Editor

The Standalone Deep Learning Editor comes in both runtime and professional version installers. It can be very useful for performing training on a client site.

The Standalone Editor can be executed using the command line with the following parameters:

```
DeepLearningEditor.exe path [--alwaysOnTop] [--disableChangingLocation] [--language "language-code"]
```

Where:

- **path** - path to the directory with the Deep Learning model,
- **--alwaysOnTop** (optional) - setting this flag prevents the Deep Learning Editor from being covered by other windows.
- **--disableChangingLocation** (optional) - the user cannot change the model path from within the editor.
- **--language** (optional) - specifies the language to be used in the editor. Possible values: "en","de","ja","pl","zh_hans","zh_hant".

The standalone editor can be started using additional HMI controls or the **Execute** filter.

6. Logging

The Deep Learning Service and Filters log some information during execution to several files, located in the *%LocalAppData%/Aurora Vision/Aurora Deep Learning 5.7* directory. Total disk space used by these files should not exceed several MB. Files older than a couple of days are automatically deleted. More information is provided in the documentation of DL_ConfigureLogging filter.

If this disk space requirement is unacceptable, the Service can be executed in "minimal logging" mode. This can be achieved by running *run_service_minimal_logging.bat* script, located at Service installation folder. Note that it will not lead to any observable performance improvement.

References

See also:

- Machine Vision Guide: Deep Learning – Deep Learning technique overview,
- Creating Deep Learning Model – how to use Deep Learning Editor.

Zebra
Aurora™ Vision

This article is valid for version 5.7.5

Legal Terms of Use Privacy Policy

ZEBRA and the stylized Zebra head are trademarks of Zebra Technologies Corp., registered in many jurisdictions worldwide. All other trademarks are the property of their respective owners. ©2026 Zebra Technologies Corp. and/or its affiliates.