



Part type designation	Part number
PLC Developer	Part of M-Base

PLC Developer

Modern automation systems solve complex tasks that go beyond the mere control of a plant. Procedural programming according to IEC 61131-3 offers the possibility to solve different tasks in an efficient and structured way. The PLC Developer supports the system programmer in all phases of software development.

Synergies in the SolutionCenter

Because of the integration of the PLC Developer in the SolutionCenter synergies can be exploited and therefore added value can be created:

Features

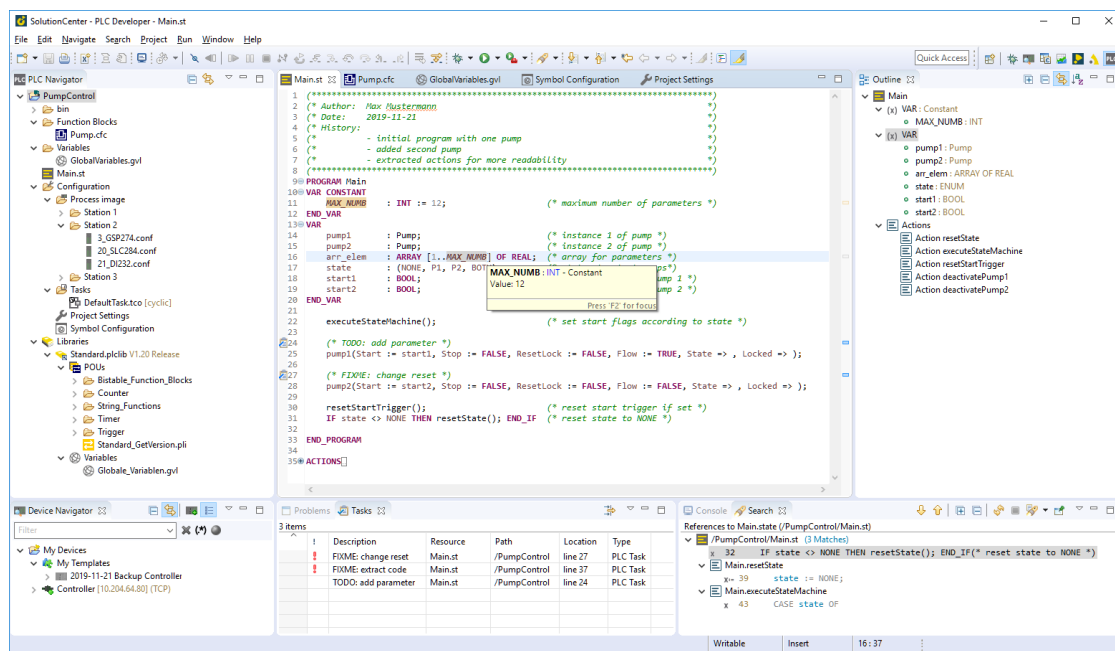
- By linking PLC projects in a Solution project with device configurations and templates, they can be combined to form a complete solution and managed as one.
- The version management function using GIT and SVN enables the user to track code modifications easily and provides the basis for a release management.
- The use of extensions, in the form of plug-ins, which can also be directly installed from the Eclipse Marketplace, simplifies project planning and programming.
- The use of SolutionCenter scripting API enables the application of modern software development methods such as continuous integration, model-driven development and test-driven development.

Efficient programming and management

The clear and intuitive editors allow a modern and efficient way of working:

- The user-defined syntax highlighting increases the readability of code in the "structured text" (ST) language.
- The modern and clear design of the user interface for programming in "Continuous Function Chart" (CFC) also permits a good overview in large programs.
- Thanks to the fully available code navigation and call hierarchy, it is also possible to easily identify interrelationships in complex applications.
- Efficient programming is particularly enhanced by the context-sensitive autocompletion and the predefined and extendable code and file templates.
- Project management in one workspace enables all changes to the source files to be stored in a local protocol. These can be called again at a later time irrespective of whether the project is version managed.

- An extensive validation function immediately notifies the programmer of programming errors when the code is edited. Quick fixes correct missing program sections such as variable declarations by adding them automatically.
- Information on interfaces and documentation of function blocks or variables can be called and displayed via tooltips without leaving the editor.
- The project-wide changing of identifiers can be carried out easily and thoroughly.
- When entering and saving the program, the notation of keywords and identifiers is corrected automatically.
- The CFC editors can be fully operated via the keyboard. This increases efficiency when creating CFC diagrams.



Modern operation through modular structure

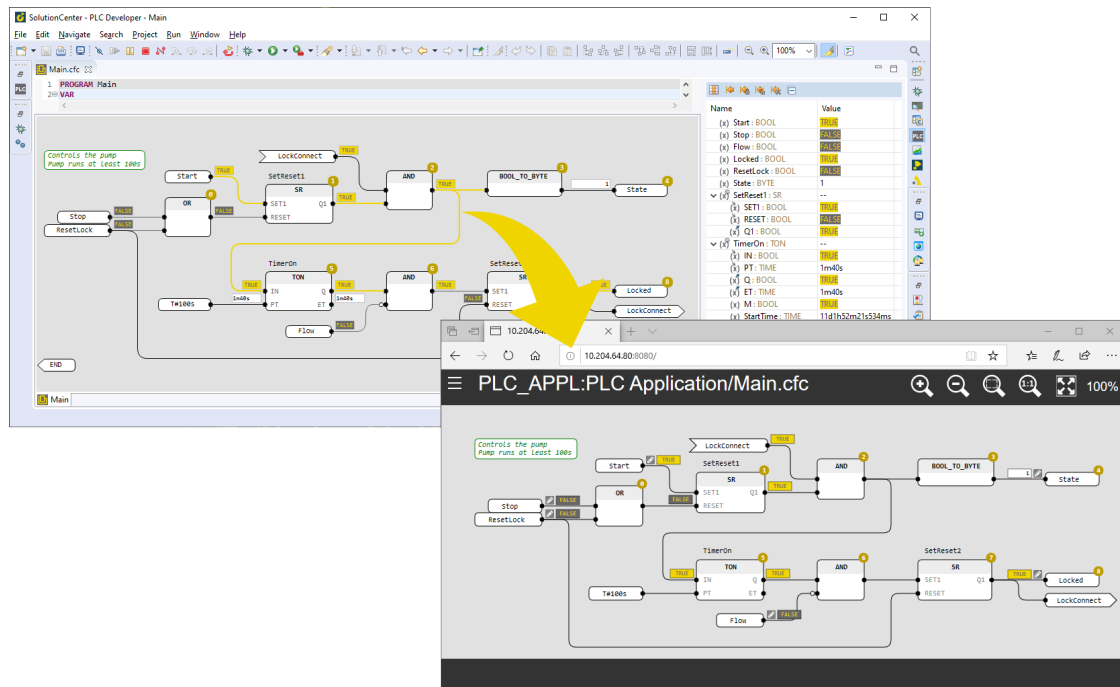
The PLC Developer is open, flexible and transparent designed to take engineering to a new level:

- All sections of the source code are saved in text form in the project in a defined file format, which opens the possibility for the user to generate modules and configurations. Recurring operating steps can thus be automated and copy/paste errors prevented.
- The structuring of the program sections can be selected as required and thus enables the application program to have a modular design.
- In order to ensure a reproducible build, the libraries used are saved in the project. An update produces a clear display of the libraries present in a newer version.
- Creation and debugging of user-defined libraries in ST, CFC, C and C++ languages is possible throughout and particularly supports the reusability of code in other projects.
- Programming is carried out irrespective of the target platform, which only has to be selected at the time of execution. After compilation, the created application program can be transferred directly to an M200 controller and run.
- Externally visible variables can be configured clearly in the symbol configuration.

Better quality through rapid troubleshooting

The debug framework of the PLC Developer is extremely helpful during the development of new application programs and to find errors:

- After a debug session is started, the actual variable values can be monitored without stopping the application program. This can be carried out via the clearly designed monitoring table in ST or the inline monitoring in CFC. The individual variables and structures of the application program can also be combined in variable lists and monitored jointly, plotted in a graph and set.
- Activating the flow control makes it possible to follow the program flow without stopping the processing.
- An application program can be stopped at any position via a breakpoint and then executed in steps. Breakpoints can be managed jointly and activated or deactivated altogether.
- When the application program is stopped, the stack frame is displayed by which it is also possible to navigate to the functions called.
- The dynamics of variables can be better assessed by highlighting variable values when they change.
- Values of boolean variables are highlighted in color in order to better identify the current status of an application program.
- Code modifications can be applied online without stopping the application program. The PLC Developer detects during the transfer whether the online change can be carried out.
- It is possible to debug several application programs simultaneously. When an application program with several tasks is debugged, the active debug task can be selected when connecting and changed during the debug session.
- PLC Insight is an automatically generated webMI visualization based on selected CFC blocks. It provides users with access to the application logic without the need for an engineering tool. The visualization is installed on the target controller together with the application program. Along with the signal flow, values of exported variables can also be monitored and changed.



PLC Developer

General

Integration	<ul style="list-style-type: none"> Management of several PLC projects in one workspace Management of all devices and PLC projects in one workspace Linking of PLC projects in the Solution projects for joint management
Display	<ul style="list-style-type: none"> Predefined perspectives with views for processing PLC projects Clear, freely selectable arrangement of views and editors
Revision control	<ul style="list-style-type: none"> GIT (local and remote) ZVN (local and remote)
Libraries	<ul style="list-style-type: none"> Integration of libraries as copies with version comparison Version display Integration of standard libraries Integration of PLC libraries Integration of external PLC libraries (C/C++, MATLAB®/Simulink®)
Automation	<ul style="list-style-type: none"> Running scripts in the SolutionCenter (JavaScript, Python) Predefined modules for creating and transferring application programs
Expandability	<ul style="list-style-type: none"> Extensions (plug-ins) installable via Eclipse Marketplace Use of user-created Eclipse plug-ins
Scalability	<ul style="list-style-type: none"> Creation of the application programs independent of the target platform Transfer and execution on all M200 CPUs
Structure	<ul style="list-style-type: none"> Combination of projects in working sets Folder structure freely definable in the project Full access to all source code files
Compatibility	<ul style="list-style-type: none"> Compatible with M-PLC (compiler and runtime) Import of M-PLC projects possible

Editors

Languages	<ul style="list-style-type: none"> Structured text (ST) Continuous function chart (CFC) Can be combined as required
-----------	--

Editors	
Code navigation	<ul style="list-style-type: none"> Find references (in the project) Call hierarchy (up to the task) Caller hierarchy (hierarchy levels expandable) Opening declaration (local and global)
Auto-complete	<ul style="list-style-type: none"> Context-sensitive (declaration, call, program section) Predefined source code templates User-defined source code templates
Autocorrection	<ul style="list-style-type: none"> Correction of identifier notation Correction of keyword notation
Syntax highlighting	<ul style="list-style-type: none"> Freely configurable Separate for structured text, global variable lists, data types and hardware configuration
Quick fixes	<ul style="list-style-type: none"> Declaration of variables and function block instances Creation of missing elements (connection marks, jumps and jump labels) Updating of the function block interface
Change tracking	<ul style="list-style-type: none"> Comparison with locally stored versions (local protocol) Comparison with source code from SVN or GIT
Changes	<ul style="list-style-type: none"> Project-wide renaming of identifiers
Information	<ul style="list-style-type: none"> Context-sensitive information in the tooltip on identifier (ST) Context-sensitive information on function block and pin (CFC)
Keyboard operation	<ul style="list-style-type: none"> Execution of most frequently used actions with keyboard shortcuts Programming possible also in CFC via keyboard
Validation	<ul style="list-style-type: none"> Syntactical and semantic checking of the source code on typing Display of problems directly in the source code and in the error view
Configuration	
Symbol configuration	<ul style="list-style-type: none"> Visibility of the local and global variables Access to local and global variables Export of collective, structure and array entries Inheritable configurations
Global variables lists	<ul style="list-style-type: none"> Declaration of global variables Declaration of global constants Declaration of global retain variables
Variable configuration	<ul style="list-style-type: none"> Assignment of variables to physical hardware addresses
Task configuration	<ul style="list-style-type: none"> Up to 16 tasks configurable Configuration of called programs Configuration of task type (cyclic, sync, event/error interrupt, free wheeling) PTP synchronization Watchdog timeout
Process image	<ul style="list-style-type: none"> Hardware import of online and offline device (selective) Manual addition of hardware modules
Project settings	<ul style="list-style-type: none"> Version number Module name (m-file) Error tolerance Multicore suitability Conflict handling with retain variables Memory layout Marker settings

Online Operation	
Installation	<ul style="list-style-type: none"> • Online change for code changes possible • Target controller permanently or dynamically selectable • Installation on target controller temporarily or permanently • Name of the module instance freely definable
Initialization	<ul style="list-style-type: none"> • Selectable debug task on start (with multitasking projects)
Deploy configurations	<ul style="list-style-type: none"> • Several configurations per project possible • Combined execution of several configurations possible (launch groups) • Management of favorites
Debugging - Troubleshooting	
Stepwise processing	<ul style="list-style-type: none"> • Stopping of the application program via breakpoint • Step into/over/return
Monitoring	<ul style="list-style-type: none"> • Variable values in the monitoring table in the editor (ST/CFC) • Variable values inline (only CFC) • Variable list for selected variables • Display of executed codes via flow control
Display	<ul style="list-style-type: none"> • Changing the format of INT values (dec, hex, bin) • Current stack frame when the application program is stopped
Libraries	<ul style="list-style-type: none"> • Internal libraries by incorporating the source codes (linked resources) • External libraries by connecting with the source codes (C/C++)
Manipulation	<ul style="list-style-type: none"> • Variable values can be set (write once) • Variable values can be forced (writing before each execution)
Visualization	<ul style="list-style-type: none"> • Generation of a complete webMI visualization from CFC blocks (PLC Insight) • Transfer of the visualization to the target controller together with the application program • Monitoring and changing the exported variables via the visualization • Tracking the signal flow based on the connections between the function blocks