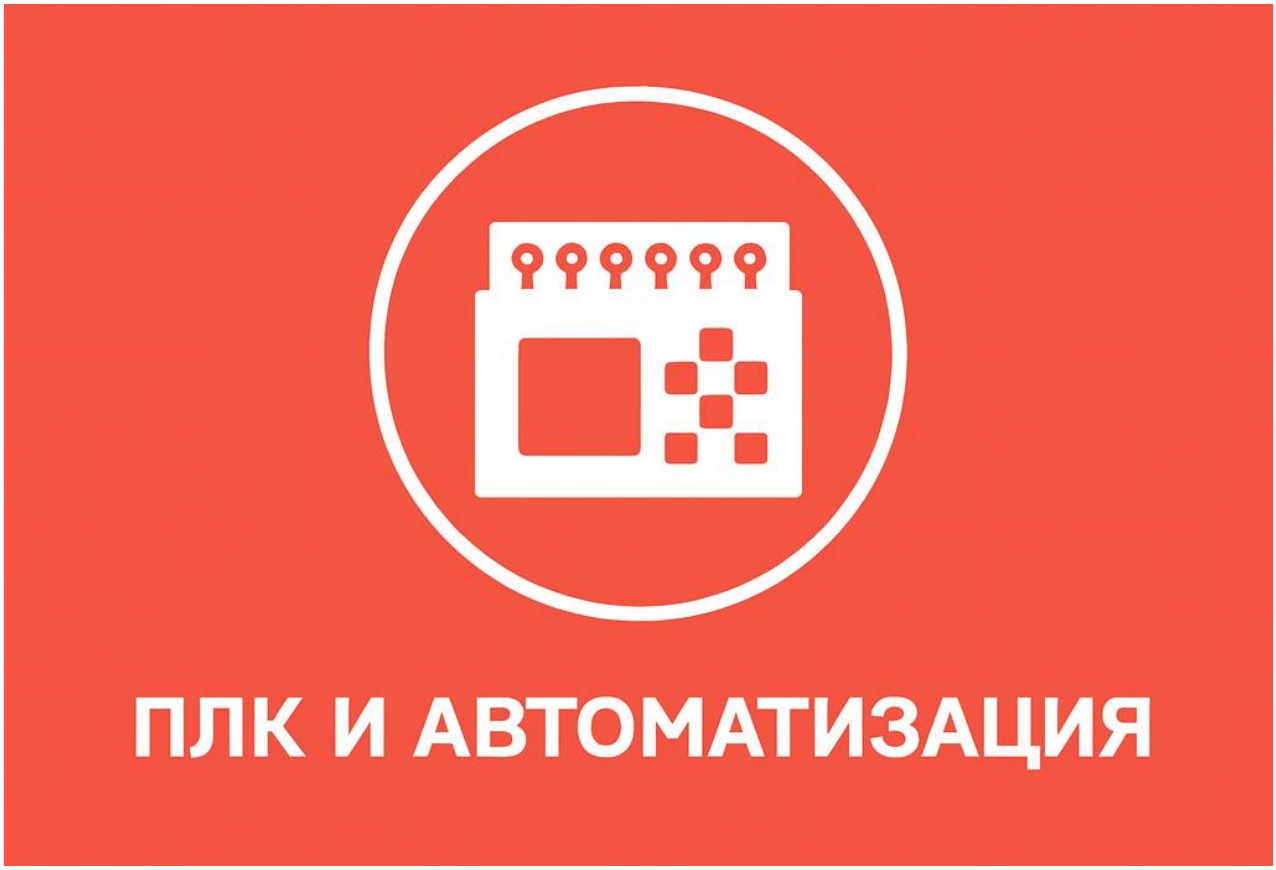

Какие лучшие практики ST рекомендует IEC 61131-3

Telegram: ПЛК и автоматизация

Электронные учебные пособия по автоматизации и программированию ПЛК

Это развёрнутый практический гайд по лучшим практикам Structured Text (ST) в духе IEC 61131-3 и PLCopen. Он рассчитан на опытного инженера, который хочет писать промышленный код «по-стандартному» и масштабируемо.

1. Архитектура и модульность: как правильно «разрезать» систему

IEC 61131-3 вводит чёткую модель **POU (Program Organization Units)**: Program, Function Block, Function. Это не просто формальность, а основа архитектуры.

1.1. Program, Function Block, Function — роли и границы

- **Program**

  Верхний уровень логики, которая привязана к конфигурации ПЛК: задачи (tasks), циклы, привязка к оборудованию.

  Практика:

  - Program отвечает за «оркестрацию»: вызовы FB, координацию подсистем.
  - В Program не должно быть «мяса» алгоритмов — только композиция.

- **Function Block (FB)**

  Объект с **собственной памятью** (VAR/VAR_STAT/RETAIN) и интерфейсом (VAR_INPUT/OUTPUT/IN_OUT).

  FB нужен, когда:

  - Поведение зависит не только от текущих входов, но и от прошлого состояния (TON, PID, шаговые алгоритмы).
  - Нужно многократно использовать один и тот же алгоритм с разными экземплярами (насосы, клапаны, оси).

- **Function (FUN)**

  Чистая функция: нет внутреннего состояния, один выход, результат зависит только от входов.

  Использование:

  - Математика, преобразования единиц, конверсия типов.
  - Локальные вычислительные подфункции, где важна предсказуемость и отсутствие сайд-эффектов.

Правило:

- Нужна память между циклами — **FB**.
- Нужен «чистый» расчёт — **Function**.
- Нужен верхний уровень организации — **Program**.

1.2. Модульная декомпозиция и архитектура

IEC 61131-3 и практические гайды рекомендуют модульную архитектуру:

- Делайте **FB-уровень «device»**:
  - FB_Valve, FB_Motor, FB_Axis, FB_Robot, FB_Conveyor и т.п.
  - Каждый FB инкапсулирует логику управления конкретным устройством (входы: команды, выходы: состояния, ошибки).
- Поверх device-FB делайте **process-FB**:
  - FB_TankControl, FB_PackLine, FB_MixingUnit.
  - Они используют device-FB как «поддетали».

- Program только связывает:
  MainProgram вызывает FB_TankControl, FB_PackLine, управляет режимами/рецептами.

Плюсы:

- Стандартизация: библиотека типовых блоков.
- Параллельная разработка: разные инженеры работают над разными FB.
- Сопровождаемость: чтобы использовать FB, достаточно знать интерфейс, а не внутренности.

2. Именование и стиль: чтобы код читали без боли

Стандарт не заставляет выбирать конкретный стиль, но IEC 61131-3 и PLCopen сильно рекомендуют **строгие соглашения об именах**.

2.1. Префиксы для типов и экземпляров

Пример (Beckhoff/PLCopen-совместимый стиль):

Типы (TYPE):

Объект	Префикс	Пример
FUNCTION_BLOCK	FB_	FB_TankControl
FUNCTION	F_	F_CalcPressure
STRUCT	ST_	ST_TankStatus
ENUM	E_	E_Mode

PROGRAM	P_	P_Main
TYPE (alias)	T_	T_TemperatureDegC

Экземпляры:

Объект	Префикс	Пример
FB	fb	fbTank1
STRUCT	st	stTank1Status
ENUM	e	eSystemMode

Рекомендованный стиль:

- Префикс типа — **верхний регистр + подчёркивание** (FB_TankControl).
- Префикс экземпляра — **нижний регистр** (fbTankControl).
- CamelCase для составных имен (TankHighLevelAlarm).
- Не использовать пробелы и дополнительные подчёркивания внутри имени (кроме разделения префикса и имени).

2.2. Имена переменных

Хорошая практика — сразу кодировать семантику в имени:

- Тип сигнала:
  - `b` — BOOL (`bPumpStartCmd`, `bTankLevelOk`).
  - `r` — REAL (`rTankLevel`, `rFlowRate`).

    - di — DINT, ui — UINT и т.д.
- Роль:
    - Cmd, Req — команды/запросы.
    - Sts, Stat — статусы.
    - Err, Alm — ошибки/аварии.

Плохо: x1, value, temp.

Хорошо: rTankLevelSetpoint, bMotorReady, eSystemMode.

Это облегчает чтение, статический анализ и поиск по проекту.

3. Области видимости и память: VAR, VAR_INPUT, VAR_IN_OUT, RETAIN

IEC 61131-3 строго определяет **видимость и время жизни переменных**.

3.1. Основные секции

- **VAR_INPUT**

  «Параметры по значению» — входы POU.

    - Не следует записывать в VAR_INPUT внутри блока (некоторые среды разрешают, но это считается плохим стилем).
    - Если нужен модифицируемый параметр — используйте VAR_IN_OUT.
- **VAR_OUTPUT**

  Выходные значения FB/FUN/Program. Значение образуется на основе входов и внутреннего состояния.
- **VAR_IN_OUT**

  Передача по ссылке, фактически аналог указателя/ссылки.

- Когда POU должен **мутировать** внешний объект (например, заполнить структуру конфигурации или изменить состояние внешней переменной).
- Аккуратно: изменения видны вызывающей стороне.

- **VAR**

  Локальные статические переменные блока — их состояние сохраняется между циклами вызова.

- **VAR_GLOBAL/VAR_EXTERNAL**

  Глобальные переменные. Использовать минимально, только там, где реально нужен глобальный контекст (например, системные флаги).

- **CONSTANT**

  Именованные константы для философии «никаких magic numbers».

- **RETAIN**

  Переменные, сохраняющие значение при отключении питания (в энергонезависимой памяти).

  - RETAIN имеет смысл только для «настоящих» технологических параметров: счётчики циклов, настройки пользователя, накопленные значения.
  - Нельзя злоупотреблять — память ограничена, возможна деградация флеш.

3.2. Временные и статические переменные (расширения языков)

Многие среды (TwinCAT, Schneider и др.) добавляют:

- **VAR_TEMP** — временные локальные переменные:

- Инициализируются заново при каждом вызове FB/Program.
- Не сохраняются между циклами.
- Используются как «локальные переменные стека».

- **VAR_STAT** — статические переменные в FUN/FB:
  - Инициализируются один раз при первом вызове и живут весь срок службы экземпляра.
  - Удобны для реализации внутреннего состояния в функциях.

Практика:

- Если значение не нужно сохранять — используйте VAR_TEMP (экономия памяти и меньше когнитивной нагрузки).
- Если нужно скрытое persistent-состояние — VAR_STAT.

4. Структурирование данных: ARRAY, STRUCT, ENUM, REFERENCES

Одна из сильных сторон IEC 61131-3 — богатая система типов.

4.1. Массивы (ARRAY)

Массивы с произвольными диапазонами индексов:

```
TYPE
   ANALOG_16_INPUT_DATA : ARRAY[1..16] OF INT;
END_TYPE
```

Практика:

- Используйте осмысленные диапазоны ([1..N] или [E_EnumType]) вместо «0..N-1», если так удобнее для технолога.
- Вводите типы массивов (как T_ или отдельный TYPE), а не «голые» ARRAY везде.

4.2. Структуры (STRUCT)

Структура объединяет различные типы в один логический объект:

```
TYPE ST_MotorDrive :
STRUCT
   eState       : E_MotorDriveState;
   rCurrent     : REAL;
   udStartCount  : UDINT;
   tTotalRunTime : TIME;
END_STRUCT
END_TYPE
```

Использование:

- Описывайте **device status** через STRUCT (FB_Valve.Status, FB_Motor.Status и т.п.).
- Передавайте сложные объекты через VAR_IN_OUT или VAR_INPUT/OUTPUT как единое целое.

4.3. Перечислимые типы (ENUM)

ENUM повышает читаемость и защищает от ошибок:

```
TYPE
   E_AnalogSignalType : (SingleEnded, Differential);
END_TYPE
```

Не делайте: `Mode : INT; IF Mode = 3 THEN ...`

Делайте: `eMode : E_OperatingMode; IF eMode = E_OperatingMode.Auto THEN ...`

5. Стиль ST-кода: форматирование, комментарии, магические числа

5.1. Форматирование и отступы

Хотя синтаксис ST не зависит от пробелов и регистра, читаемость критична.

Рекомендации:

- 2–4 пробела отступа, без табов (или строго единый стиль в проекте).
- Каждый оператор — с новой строки.
- Вложенные IF/CASE ровняйте по уровню вложенности.

Пример читаемого фрагмента:

```
IF bStartCmd AND NOT bError THEN
   IF rLevel < rLevelMin THEN
      eState := E_State.LowLevel;
   ELSIF rLevel > rLevelMax THEN
      eState := E_State.HighLevel;
   ELSE
      eState := E_State.Normal;
   END_IF;
```

```
END_IF;
```

5.2. Комментарии

Комментарии в ST:

- // — однострочный.
- (* ... *) — блочный.

Принципы:

- Комментарий объясняет **зачем**, а не «что делает строка» (это видно из кода).
- Каждый FB должен иметь шапку: назначение, автор, версия, важные детали реализации.
- Сложные участки алгоритма сопровождайте пояснением.

6. Управляющие конструкции: IF, CASE, циклы

6.1. IF vs CASE

IEC 61131-3 определяет полный набор конструкций IF/ELSIF/ELSE и CASE OF.

Практика:

- Для множественного выбора по состоянию/режиму используйте **CASE**, а не длинную цепочку IF/ELSIF.
- ENUM + CASE — базовый паттерн для state-machine:

```
CASE eState OF
  E_State.Idle:
```

```
    FB_Motor.bStart := FALSE;
  E_State.Starting:
    FB_Motor.bStart := TRUE;
  E_State.Running:
    // ...
  E_State.Error:
    // ...
END_CASE;
```

6.2. Циклы: FOR / WHILE / REPEAT

Циклы следует использовать **с особой осторожностью** в ПЛК, чтобы не нарушить требование к циклу сканирования.

- **FOR** — предпочитается, когда заранее известен диапазон итераций (перебор массива сигналов).
- **WHILE / REPEAT** — использовать только для ограниченных по числу шагов задач, чтобы избежать «вечных» циклов.

Практика:

- Всегда иметь явный лимит итераций для любых циклов.
- Избегать циклов, выполняющих «до события» без предела.

7. Обработка ошибок и безопасность (Safety-ориентированный ST)

Для safety-критичных приложений IEC 61131-3 и PLCopen Safety дают дополнительные рекомендации.

7.1. Структура обработки ошибок

Рекомендуется силами ST реализовывать:

- Единый формат ошибок:
  - `bError` (BOOL)
  - `iErrorId` (INT/DINT)
  - `sErrorMsg` (STRING)
- Чёткое различение:
  - Ошибки входов (датчик неисправен).
  - Логические ошибки (конфликт команд).
  - Системные ошибки (таймауты, внутренние исключения).

Пример шаблона в FB:

```
IF rTankLevel > rMaxLevel THEN
   bError   := TRUE;
   iErrorId := 1001;
   sErrorMsg := 'Tank level high';
END_IF;
```

7.2. Ограничения в safety-коде

PLCopen Safety указывает, что для безопасных частей приложения рекомендуется осторожное использование мощных конструкций ST (FOR, IF..ELSE, CASE), чтобы избежать трудно проверяемой логики.

Практика:

- Safety-логика часто реализуется более «плоско» и явно, даже ценой повторов.
- Код safety-части должен легко прослеживаться до FMEA/SIL-документов.

8. Тестирование, сложность и рефакторинг ST-кода

8.1. Контроль сложности

Исследования по IEC 61131-3 предлагают набор метрик сложности (размер, структура данных, поток управления, информационный поток, лексическая структура).
Опрос промышленных экспертов показал, что **Cognitive Complexity** лучше всего отражает фактическую «трудность понимания» ST-кода.

Практические выводы:

- Длинные функции/FB с высокой вложенностью IF/CASE нужно дробить.
- Минимизировать вложенность более 3–4 уровней.
- Выносить сложные вычисления в отдельные FUN.

8.2. Тестирование и валидация

Рекомендованные методы для ST-кода:

- **Unit-тесты** для FB/FUN — проверка логики независимых блоков.
- **Интеграционные тесты** — проверка взаимодействия FB на уровне subsystem.
- **Моделирование и симуляция** — максимально использовать симуляторы ПЛК и цифровые двойники.

- **Верификация** — для критичных систем применять подходы model checking и спецификационно-ориентированную проверку.

9. Объектно-ориентированный ST (3-я редакция IEC 61131-3)

Начиная с 3-й редакции IEC 61131-3 (2013), ST получил полноценные ООП-механизмы.

9.1. Классы, методы и интерфейсы

Официально добавлены:

- **Классы** и **методы** — реализация поведения в объектном стиле.
- **Интерфейсы** — контракты, которые реализуют классы/FB.
- **Наследование** — возможность создавать иерархии.

Практика:

- Создавайте базовые абстрактные FB/классы для семейств устройств (MotorBase, ValveBase).
- Наследуйтесь для конкретных реализаций (FB_MotorVFD, FB_MotorSoftStarter).
- Используйте интерфейсы для общих контрактов (IMotor, IAxis).

ОО-подход позволяет ещё больше усилить модульность и стандартизацию библиотек.

10. Итоговый чек-лист best practices ST по IEC 61131-3

Для готового/рефакторимного проекта можно пройтись по чек-листу:

1. **Архитектура**
    - Есть ли чёткое разделение Program / FB / FUN?
    - Логика устройств инкапсулирована в FB?
    - Есть ли библиотека типовых FB (клапан, насос, ось)?
2. **Именование**
    - Используются ли префиксы FB_/F_/ST_/E_/T_ и fb/st/e на экземплярах?
    - Имена переменных отражают тип и смысл (b, r, i, e, Cmd/Stat/Err)?
3. **Переменные и память**
    - VAR_INPUT/OUTPUT/IN_OUT используются по назначению?
    - Глобальные переменные минимизированы?
    - RETAIN/CONSTANT применяются осмысленно?
4. **Структуры данных**
    - Используются STRUCT и ENUM вместо «голых» наборов переменных и магических чисел?
    - Массивы имеют осмысленные диапазоны?
5. **Стиль и читаемость**
    - Код форматирован, нет «лестниц» из IF без CASE?
    - Комментарии описывают намерение, а не очевидное?
    - Нет жёстко зашитых чисел без констант?
6. **Безопасность и ошибки**

- Есть единый формат ошибок в FB (bError, ErrorId, ErrorMsg)?
- Safety-код написан «прозрачно» и отслеживается до требований SIL?

7. **Сложность и тесты**
   - Длинные FB/Program разбиты, вложенность ограничена?
   - Есть unit-тесты/симуляции для ключевых блоков?