

Architectural Definition Report: Industrial HMI & Motion Control for Phase 5 Post-Processing

1. Executive Summary & System Architecture

1.1 Project Context and Scope

This report serves as the definitive architectural blueprint for Phase 5 of the Industrial HMI & Motion Control System project. Phase 5, designated "Post-Processing & Code Generation," represents the critical bridge between the theoretical, geometric validation of Phase 4 and the deterministic, physical execution of the machine cycle. The input for this phase is a fully validated BendSequence object—a high-level computational representation of the manufacturing process containing a sequential list of MachineAction entities (Bend, Rotate, ABA, Reposition) that have been rigorously checked for collisions and physics feasibility.

The primary mandate of this architecture is to transform this static geometric intent into a dynamic, real-time control strategy that drives a Beckhoff/TwinCAT PC-based controller. This transformation involves two distinct but synchronized outputs: a proprietary "Machine Instruction Set" (PMIS) for the real-time controller and a high-fidelity, interactive 3D simulation structure for the "FACE" operator interface (HMI). A defining challenge of this architecture is the integration of "MAC 2.0" (Machine Automatic Correction)—an adaptive control technology that requires the instruction set to support conditional branching logic based on real-time sensor feedback, essentially elevating the NC code from a linear script to a logic-driven program.¹

Furthermore, the architecture must address the computational constraints of the HMI platform. Industrial HMIs often run on hardware shared with the control runtime or on limited embedded PCs. Therefore, the visualization pipeline must achieve "Lightweight Playback," avoiding the prohibitively expensive Constructive Solid Geometry (CSG) operations typically used in CAD kernels. Instead, we propose a "Skeletal Bending" methodology that leverages GPU-accelerated vertex skinning.³ Finally, the system must close the loop between the virtual and physical worlds by implementing a "Digital Twin" view that overlays real-time EtherCAT position feedback onto the target simulation, facilitating intuitive "Touch-to-Code" debugging for the machine operator.⁵

1.2 Hardware and Software Topology

The target system is a PC-Based Controller utilizing the Beckhoff TwinCAT 3 automation suite. This topology dictates specific architectural constraints and opportunities regarding data

exchange, real-time performance, and connectivity.

The central processing unit is an Industrial PC (IPC), likely a multi-core CX series or a C60xx ultra-compact IPC, running Windows 10 IoT Enterprise. TwinCAT 3 utilizes CPU isolation to reserve specific cores for the Real-Time Kernel (XAR), while the remaining cores service the Windows Operating System and the .NET-based HMI. This separation allows for a highly responsive HMI (WPF) that communicates with the machine logic via the Automation Device Specification (ADS) protocol, a message-based middleware that operates over the internal virtual network.⁶

Table 1: System Layer Definition

Layer	Component	Function	Technology Stack	Cycle Time
L3: Visualization	"FACE" HMI	3D Rendering, Operator Interaction, Job Loading	C#.NET 6, WPF, Helix Toolkit (DirectX)	~16ms (60 Hz)
L2: Middleware	ADS Router	Data Marshaling, Event Notification, Buffer Management	Beckhoff ADS, TcAdsClient	Asynchronous
L1: Logic Control	PLC Runtime	State Machines, Instruction Interpreter, Safety Logic	TwinCAT 3 PLC (IEC 61131-3 ST)	1-10ms
L0: Motion Control	NC Task	Trajectory Generation, Kinematics, Servo Loops	TwinCAT NC I / CNC	250μs - 2ms
Fieldbus	EtherCAT	Drive communication, Sensor I/O	EtherCAT, CoE (CANopen over EtherCAT)	Synchronous

		(LAMS)		
--	--	--------	--	--

The architecture leverages the concept of "Software CNC." Unlike traditional hardware-based CNCs (like Fanuc or Siemens generic controllers), TwinCAT allows us to write a custom interpreter in Structured Text (ST) or C++ that runs directly in the real-time context.⁵ This is crucial for implementing the proprietary K-Flow instruction set defined in this report, as it permits deep integration of non-standard logic (like the MAC 2.0 adaptive bending loops) directly into the motion planner.

1.3 The Necessity of a Proprietary Format

Standard G-code (DIN 66025) is historically rooted in subtractive manufacturing (milling/turning), where the primary operation is moving a tool tip along a vector path. Sheet metal bending, however, is fundamentally different. It involves managing the deformation of the workpiece, handling complex tool kinematics (like the Automated Bend Angle or ABA), and managing material properties like springback.⁹

While G-code can move axes (e.g., G01 Y-100), it lacks the semantic richness to describe *intent*. For example, a "Bend" operation is not just a Y-axis move; it is a sequence of: Fast Approach -> Mute Point -> Pressing -> Decompression -> Retraction. If we were to compile this into raw G-code in the CAM phase, the resulting file would be enormous and rigid. Any adjustment to parameters (e.g., changing the mute point due to a sensor reading) would require re-posting the code.

By defining a proprietary object-oriented format (K-Flow), we encapsulate the *logic* of the operation. The file says <Bend Angle="90" />, and the machine controller—aware of its own current configuration and sensor inputs—calculates the precise axis moves at runtime. This "Late Binding" of trajectory generation is the key enabler for adaptive control and the robust functioning of the MAC 2.0 system.¹¹

2. Proprietary Machine Instruction Set (PMIS): "K-Flow"

The proprietary machine instruction set, herein named **K-Flow (Kinematic Flow)**, is the intermediate language that bridges the gap between the offline CAM software and the real-time machine controller. It is designed to be human-readable, extensible, and strictly typed to ensure safety and determinism.

2.1 Format Selection: XML vs. JSON vs. Binary

While JSON is popular for web applications and binary formats are efficient for storage, XML

(eXtensible Markup Language) remains the superior choice for industrial machine configurations.¹¹ The reasons are threefold:

1. **Schema Validation (XSD):** XML allows for rigorous validation against a schema (XSD) *before* the file is loaded into the machine. This prevents runtime errors caused by malformed data or missing parameters, which is critical for machine safety.¹⁴
2. **Hierarchical Structure:** Bending operations are inherently nested (e.g., A Job contains Parts, which contain Bends, which contain AxisMoves). XML represents this tree structure natively.
3. **Industrial Standard Compatibility:** Standards like PLCopen XML and PackML use XML, making it a familiar paradigm for automation engineers.¹⁵

However, parsing XML inside a real-time PLC environment is computationally expensive and non-deterministic.¹⁷ Therefore, the architecture dictates that the *parsing* happens in the HMI (non-real-time context) or a lower-priority task, while the *execution* uses a binary serialized version of the data. This "Load-Time Compilation" strategy gives us the best of both worlds: the flexibility of XML for data exchange and the speed of binary structures for execution.

2.2 K-Flow Schema Definition

The K-Flow schema is structured around the concept of a "Unit of Work." The root node is the <Job>, which contains global metadata and a sequence of <Action> nodes.

2.2.1 Header and Resources

The header encapsulates the provenance of the file and the material physics required for the adaptive algorithms.

XML

```
<KFlow version="2.0" xmlns="http://www.proprietary-machine.com/kflow">
  <Header>
    <JobID>JOB-2025-AX99-Rev3</JobID>
    <Timestamp>2025-10-27T14:30:00Z</Timestamp>
    <Generator>Phase4_PostProcessor_v1.5</Generator>
    <Material>
      <ID>SS304</ID>
      <Thickness nominal="2.0" tolerance="0.1"/>
      <TensileStrength>520</TensileStrength> <ElasticModulus>193</ElasticModulus>
    </Material>
    <SpringbackFactor>1.025</SpringbackFactor>
  </Header>
</KFlow>
```

```

</Header>
<Resources>
  <Tooling>
    <UpperTool id="UT-105" segment="FullLength" height="150.0" radius="1.0" />
    <LowerTool id="LT-220" segment="V12" width="12.0" angle="88" />
    <Manipulator id="ABA-Suction-Cup-A" />
  </Tooling>
</Resources>
<Sequence>
</Sequence>
</KFlow>

```

The <Material> block is essential for MAC 2.0. The "SpringbackFactor" is an initial guess derived from the CAM simulation. The machine will update this value at runtime based on sensor data, but it needs a starting point to plan the initial trajectory.¹⁹

2.2.2 The Action Hierarchy

Unlike linear G-code, K-Flow uses a polymorphic <Step> structure. Each step has a Type attribute that dictates the required parameters.

Table 2: K-Flow Action Types

Action Type	Description	Required Parameters	Adaptive Support
MoveAxes	Coordinated PTP motion	AxisID, TargetPos, Velocity, Accel, Jerk	No
Bend	Full bending cycle	Angle, BendLineID, CorrectionMode, MuteDist	Yes (MAC 2.0)
Rotate	Manipulator rotation (Z-axis)	Angle, LiftHeight	No
Repo	Re-gripping operation	RetractDist, ClampState	No
Probe	Sensor check cycle	SensorID, ExpectedValue,	Yes

		Tolerance	
Logic	Conditional branching	Condition, TrueStepID, FalseStepID	Yes

2.3 Implementing Conditional Logic (MAC 2.0)

The user request explicitly highlights the need for conditional logic to support MAC 2.0 corrections. In a traditional "Tape Mode" CNC, the program flows line by line. Adaptive bending breaks this paradigm: the machine must bend, measure, decide, and potentially re-bend.²¹

To support this, K-Flow implements a **State-Based XML Structure** rather than a purely sequential one. The <Bend> action is not a single command but a macro that expands into a sub-sequence based on the <AdaptiveCorrection> node.

Example: Adaptive Bend Instruction

XML

```
<Step id="20" type="Bend">
  <Description>Flange A - 90 Degree Fold</Description>
  <Kinematics>
    <TargetAngle>90.0</TargetAngle>
    <Y_Target_Theoretical>-102.5</Y_Target_Theoretical>
    <Crowning>2.5</Crowning>
  </Kinematics>

  <AdaptiveControl enabled="true">
    <Strategy>Measure_Relieve_Rebend</Strategy>
    <Sensors>
      <Sensor type="LaserTracker" side="Left" />
      <Sensor type="LaserTracker" side="Right" />
    </Sensors>
    <Logic>
      <Condition variable="MeasuredAngle" operator="GreaterThan" value="90.5">
        <CorrectionAction>
          <CalculateOffset formula="(MeasuredAngle - TargetAngle) * K_Factor" />
        </CorrectionAction>
      </Condition>
    </Logic>
  </AdaptiveControl>
</Step>
```

```
<ReBend mode="Incremental" />
</CorrectionAction>
</Condition>
</Logic>
</AdaptiveControl>
</Step>
```

When the TwinCAT interpreter encounters this block, it does not simply send a move command. Instead, it instantiates a FB_AdaptiveBend function block. This FB contains an internal state machine (CASE statement) that handles the micro-moves:

1. **Fast Down** to Mute Point.
2. **Press** to Y_Target_Theoretical.
3. **Dwell** (Decompression) to allow springback to manifest.
4. **Read Sensors** (EtherCAT LAMS).
5. **Evaluate Logic**: If MeasuredAngle > 90.5, compute offset.
6. **Correction**: Move Y-axis to Y_New.
7. **Retract**.

This encapsulation of logic within the instruction allows the CAM system to dictate *behavior* without needing to know the exact runtime conditions of the hydraulics or material batch.²⁰

3. The Real-Time Interpreter & Control Architecture

The heart of the Phase 5 execution system is the Real-Time Interpreter running on the Beckhoff IPC. This component is responsible for reading the K-Flow data and generating cyclic setpoints for the servo drives.

3.1 The "Software CNC" Approach vs. Hardware CNC

Traditional CNCs use a hardware-based interpolator that reads G-code text files. In our Beckhoff/TwinCAT architecture, we implement a "Software CNC." We utilize **TwinCAT NC I** (Interpolation) for the path generation but wrap it in a custom PLC-based sequencer.²⁴

We reject the idea of writing a parser directly in the PLC (IEC 61131-3) due to the poor string handling performance of real-time tasks. Instead, we architect a **Two-Stage Loading Process**:

1. **Stage 1: HMI Parsing (C#/I.NET)**: The HMI reads the XML file. It validates the XSD and parses the complex strings into strongly typed objects. It then serializes these objects into a flat binary array of structures (e.g., ST_MachineStep).
2. **Stage 2: ADS Block Transfer**: The HMI uses the ADS protocol to write this entire array into the PLC memory in a single block transfer. This is significantly faster than parsing

line-by-line in the controller.¹⁷

3.2 PLC Data Structure: ST_MachineAction

To support the K-Flow schema, the PLC defines a unified structure capable of holding the data for any action type. Using a generic structure with a Union or a comprehensive set of fields avoids dynamic memory allocation, which is strictly prohibited in real-time control.²⁷

Delphi

```
(* TwinCAT 3 PLC Structure Definition *)
TYPE ST_MachineAction :
STRUCT
  nStepID      : INT;
  eActionType  : E_ActionType; (* Enum: MOVE, BEND, PROBE... *)

  (* Motion Parameters *)
  fTargetPos_X : LREAL;
  fTargetPos_Y : LREAL;
  fVelocity    : LREAL;

  (* Adaptive Logic Parameters *)
  bAdaptiveEnable : BOOL;
  fTargetAngle   : LREAL;
  fTolerance     : LREAL;
  eStrategy      : E_AdaptiveStrategy;

  (* Metadata for HMI Sync *)
  sDescription   : STRING(50);
  nBoneIndex    : INT; (* For 3D Visualization Linking *)
END_STRUCT
END_TYPE
```

3.3 The Sequencer State Machine

The core execution logic is a state machine implemented in Structured Text (ST). It iterates through the array of ST_MachineAction steps.

Delphi

```
(* Main Sequencer Loop - 1ms Cycle Task *)
CASE eSequenceState OF
  E_State.IDLE:
    IF bStartButton AND bJobLoaded THEN
      nStepIndex := 0;
      eSequenceState := E_State.FETCH_STEP;
    END_IF

  E_State.FETCH_STEP:
    CurrentAction := JobData;
    CASE CurrentAction.eActionType OF
      E_ActionType.MOVE:
        eSequenceState := E_State.EXECUTE_MOVE;
      E_ActionType.BEND:
        eSequenceState := E_State.EXECUTE_BEND;
    END_CASE

  E_State.EXECUTE_BEND:
    (* Call the Adaptive Bending Function Block *)
    fbAdaptiveBend(
      bExecute := TRUE,
      fTargetY := CurrentAction.fTargetPos_Y,
      fTargetAngle := CurrentAction.fTargetAngle,
      bEnableCorrection := CurrentAction.bAdaptiveEnable
    );

    IF fbAdaptiveBend.bDone THEN
      fbAdaptiveBend(bExecute := FALSE);
      nStepIndex := nStepIndex + 1;
      eSequenceState := E_State.FETCH_STEP;
    END_IF
END_CASE
```

This structure allows for the "Step-Through" debugging requested. The operator can pause the machine, and the PLC simply holds the nStepIndex. A "Jump" command from the HMI simply updates nStepIndex to a new value, and the state machine fetches the new step on the next cycle.²⁷

4. HMI Visualization Pipeline: The "FACE" UI

The "FACE" (Functional Adaptive Control Environment) interface is the operator's viewport. The architectural mandate is high-performance 3D visualization that runs smoothly alongside the heavy data processing of the machine control.

4.1 Technology Stack: WPF and Helix Toolkit

The HMI is built on **Microsoft WPF (Windows Presentation Foundation)**. While WPF provides native 3D capabilities (Viewport3D), they are insufficient for complex mechanical simulations due to poor performance with high triangle counts and lack of advanced shader support.³⁰

Therefore, we integrate **Helix Toolkit (SharpDX)**, which wraps the low-level **DirectX 11** API. This allows for hardware-accelerated rendering, instancing, and custom HLSL shaders while maintaining the ease of use of WPF for the 2D UI overlay.³²

Pipeline Architecture:

- **Rendering Thread:** Decoupled from the UI thread using `CompositionTarget.Rendering` to ensure fluid 60 FPS animation even when the UI is processing data.³⁴
- **Scene Graph:** A hierarchical tree of `SceneNode` objects representing the machine components (Frame, Beam, Backgauge, Tooling, Sheet).

4.2 Lightweight Playback: The "Skeletal Bending" Strategy

The user query asks to avoid recalculating heavy CSG (Constructive Solid Geometry) operations on the HMI. CSG involves boolean subtractions (e.g., $\text{SheetVolume} - \text{ToolVolume}$) to calculate the bent shape. This is computationally prohibitive for real-time rendering.³⁶

Architectural Decision: Skeletal Animation (Skinning) We adopt a technique from the video game industry: **Linear Blend Skinning (LBS)**. Instead of deforming the mesh geometry (vertices) on the CPU, we rig the sheet metal part with a system of virtual "Bones" and offload the deformation to the GPU Vertex Shader.³

4.2.1 The Animation Skeleton Structure

During Phase 4 (Post-Processing), the CAM software analyzes the bend lines and generates a "Rig."

1. **Root Bone:** Centered on the main face of the sheet.
2. **Joints:** Placed exactly at the bend lines (neutral axis).
3. **Leaf Bones:** Extend to the edges of the flanges.

Vertex Weighting:

The mesh vertices are "weighted" to these bones.

- Vertices on the flat flange have 100% weight to the Flange Bone.
- Vertices *inside* the bend radius have blended weights (e.g., 50% Root, 50% Flange) to create a smooth curve when the bone rotates.

4.2.2 The Export Format: GLB + AKF

Instead of exporting a sequence of meshes, the CAM software exports:

1. **Static Mesh (.GLB):** A single file containing the sheet geometry, the bone hierarchy, and the skinning weights.³⁸
2. **Animation Keyframe File (.AKF):** A lightweight map linking K-Flow Step IDs to Bone Rotation angles.
 - Step 10: Bone_1_Rot = 0.0
 - Step 20: Bone_1_Rot = 45.0
 - Step 30: Bone_1_Rot = 90.0

4.2.3 Runtime Execution

The HMI simply loads the GLB file. To animate a bend:

1. PLC sends current BendAngle via ADS.
2. HMI updates the Rotation transform of the specific Bone Node.
3. The GPU shader multiplies the vertex positions by the bone matrix.

Table 3: Performance Comparison: CSG vs. Skeletal Animation

Metric	CSG / Boolean Mesh	Skeletal Animation (Proposed)
CPU Usage	High (Mesh reconstruction every frame)	Negligible (Matrix updates only)
GPU Usage	Medium (Re-uploading vertex buffers)	High (Vertex Shader), but efficient
Memory	High (Unique mesh per frame)	Low (Single static mesh)
Visuals	"Popping" between discrete steps	Smooth, continuous interpolation
Complexity	High (Requires CAD kernel)	Low (Standard Game)

	runtime)	Engine tech)
--	----------	--------------

This approach satisfies the "Lightweight Playback" requirement perfectly. The HMI does not "calculate" the bend; it simply "poses" the existing mesh based on signal inputs.

5. Syncing with Reality: The Digital Twin Implementation

The "Digital Twin" requirement necessitates displaying the "Ghost Model" (Target Position) and the "Actual Model" (Real-time Feedback) simultaneously. This allows the operator to visually verify if the machine is executing the intended path or if there is a deviation (lag, error).

5.1 ADS Ring Buffer Strategy

Syncing visualization with a 1ms Motion Task is non-trivial. Standard Windows timers vary by 15-30ms, causing visual jitter. To map real-time EtherCAT data smoothly, we implement an **ADS Ring Buffer** mechanism.⁴⁰

1. **PLC Side:** A FastLog array fills with position data every cycle (1ms). Every 16ms (matching the 60Hz screen refresh), it marks a chunk ready for transfer.
2. **ADS Side:** The HMI utilizes AddDeviceNotification on the buffer. Beckhoff ADS acts as a "mailbox," pushing the chunk of positions to the HMI.
3. **HMI Side:** The incoming chunk is stored in a Queue. The Rendering Loop (CompositionTarget.Rendering) pulls the oldest sample from the queue and interpolates between it and the next sample.

This interpolation effectively delays the "Actual" visualization by ~20-50ms (imperceptible to the human eye) but ensures perfectly smooth motion without the "stuttering" associated with direct variable polling.³⁴

5.2 Kinematic Mapping: The "Ghost" Logic

The Ghost Model represents the *Target*. Its position is derived directly from the currently active ST_MachineAction in the K-Flow sequence.

Rendering Layers:

- **Layer A (Reality):** The solid, metallic-shaded machine model driven by Axis.ActPos from EtherCAT.
- **Layer B (Ghost):** A semi-transparent (Alpha = 0.3), wireframe or blue-tinted model

driven by Axis.SetPos (Target).

Synchronization Logic:

In the HMI's render loop:

C#

```
// Pseudo-code for Digital Twin Loop
void OnRender(object sender, EventArgs e)
{
    // 1. Update Reality (From Ring Buffer)
    var actualPos = buffer.GetInterpolatedSample(hiResTimer.Now);
    RealModel.Transform = new TranslateTransform3D(actualPos.X, actualPos.Y, actualPos.Z);

    // 2. Update Ghost (From Current Step Target)
    var currentStep = JobManager.CurrentStep;
    GhostModel.Transform = new TranslateTransform3D(currentStep.TargetX,
currentStep.TargetY, currentStep.TargetZ);

    // 3. Update Sheet Metal Bones
    // Map linear Y-axis position to angular Bone rotation using Air-Bend Formula
    double bendAngle = Kinematics.CalculateAngleFromY(actualPos.Y);
    SheetSkeleton.Bones.Rotation = bendAngle;
}
```

The Kinematics.CalculateAngleFromY function is a crucial piece of the Digital Twin. It duplicates the machine's internal geometric model on the HMI side, allowing the visualization to translate linear ram depth (which is what the machine knows) into a bend angle (which is what the operator sees).²³

6. Operator Interaction: "Touch-to-Code" Debugging

The request specifies a feature where the operator can touch a specific bend line on the 3D model to jump to that line in the code. This requires a bidirectional mapping system:

Geometric Picking -> Logic Association.

6.1 Raycasting and Hit Testing

In a 3D viewport, a "touch" is a 2D coordinate (u, v) on the screen. To find what 3D object is under that point, we use **Raycasting**.⁴⁴

1. **Ray Generation:** A ray is projected from the camera's near plane through the (u, v) coordinate into world space.
2. **Intersection Test:** The Helix Toolkit performs an intersection test against the bounding boxes (Octrees) of the scene meshes.⁴⁶
3. **Primitive Identification:** The hit test returns the specific MeshGeometry and the TriangleIndex that was hit.

6.2 The "Bone Map" Metadata

Standard hit testing tells you "You hit the Sheet Metal Mesh." It does not tell you "You hit Bend Line 3." To solve this, we leverage the **Skeletal Rigging** defined in Section 4.2.

Since the mesh is skinned, every vertex is associated with a Bone.

1. **Retrieve Hit Triangle:** Get the vertices v_1, v_2, v_3 of the hit triangle.
2. **Query Weights:** Check the BoneWeights array for these vertices.
3. **Identify Bone:** If the vertices are weighted > 0.5 to Bone_3, then the user touched the area controlled by Bone_3.

6.3 Reverse Lookup Logic

The CAM export included metadata linking Bones to Steps.

- **Lookup Table:** Dictionary<BoneID, StepID> loaded from the K-Flow header.
- **Action:**
 1. User touches visual bend.
 2. Raycast identifies Bone_3.
 3. Lookup finds Bone_3 -> Step_ID_45.
 4. **HMI Logic:**
 - Scroll the Code Viewport to Line 45.
 - Highlight the line in yellow.
 - If the machine is in "Setup" mode, send ADS Command:
Write("MAIN.Sequencer.nNextStep", 45).

This feature transforms the 3D view from a passive display into an active navigation interface, significantly speeding up setup and debugging times for complex parts.

7. Implementation Roadmap and Conclusion

7.1 Detailed Roadmap

1. **Phase 5.1: Schema Definition (Weeks 1-2):** Finalize the K-Flow XSD. Write the C# serializer for the CAM Post-Processor.
2. **Phase 5.2: PLC Interpreter (Weeks 3-6):** Implement the ST_MachineAction structure and the generic State Machine in TwinCAT. Test with dummy data.
3. **Phase 5.3: HMI Core (Weeks 4-8):** Initialize the Helix Toolkit Viewport3DX. Implement the GLB loader and the Skeletal Animation controller.
4. **Phase 5.4: ADS & Sync (Weeks 8-10):** Implement the Ring Buffer in PLC and the consuming queue in C#. Tune the interpolation for smooth 60 FPS rendering.
5. **Phase 5.5: Interaction (Weeks 10-12):** Implement Raycasting and the "Touch-to-Code" reverse lookup.

7.2 Conclusion

The architecture defined in this report successfully addresses the complex requirements of Phase 5. By adopting the **K-Flow** XML-based instruction set, we enable the conditional logic required for **MAC 2.0** without sacrificing the safety of strict schema validation. The **Skeletal Animation** strategy for the HMI solves the critical "Lightweight Playback" constraint, allowing high-fidelity simulation without CSG overhead. Finally, the **Digital Twin** pipeline, powered by ADS Ring Buffers and Kinematic Mapping, ensures that the operator's view is a strictly synchronized, interactive window into the machine's physical reality. This system represents a significant leap forward from legacy G-code controllers, positioning the machine for the Industry 4.0 era of adaptive, data-driven manufacturing.

8. Appendix: Detailed Data Tables

Table 4: ADS Communication Throughput Estimates

Data Type	Frequency	Payload Size	Bandwidth	Strategy
Status Monitoring	10 Hz	~200 Bytes	2 KB/s	OnValueChange Notification
Real-time Positions (Digital Twin)	1000 Hz (Buffered)	~300 Bytes (50 samples)	300 KB/s	Ring Buffer / Cyclic Notification
Job Data (K-Flow)	Once (Load)	1-5 MB	N/A	Large Block Write

				(WriteAny)
Commands (Start/Stop)	Event	1-4 Bytes	Negligible	Direct Sync Write

Table 5: Conditional Logic Operators for K-Flow

Operator	XML Representation	PLC Logic Equivalent	Use Case
Greater Than	<Condition op="GT">	IF val > limit	Checking sensor angle vs target
In Range	<Condition op="InRange">	IF val >= min AND val <= max	Verifying tolerance compliance
Timeout	<Condition op="Timeout">	IF timer.Q	Sensor failure handling
Step Complete	<Condition op="StepDone">	IF step.Done	Sequential progression

Works cited

1. Salvagnini P2 #2 Under the Hood - YouTube, accessed February 4, 2026, <https://www.youtube.com/watch?v=YEim9UkEvAk>
2. #1 Salvagnini S4 + P4 line FACE Software - YouTube, accessed February 4, 2026, <https://www.youtube.com/watch?v=IsmBVhUAZSM>
3. Skeletal Animation: A Comprehensive Guide - GarageFarm, accessed February 4, 2026, <https://garagefarm.net/blog/skeletal-animation-a-comprehensive-guide>
4. Implementing Skeletal Animation - Andrea Venuta, accessed February 4, 2026, <https://veeenu.github.io/blog/implementing-skeletal-animation/>
5. Intuitive CNC HMIs with simulation functionality | Beckhoff USA, accessed February 4, 2026, <https://www.beckhoff.com/en-us/company/press/twincat-hmi-components-for-cnc-specific-user-interfaces-2025-09.html>
6. TC1000 | TwinCAT 3 ADS | Beckhoff USA, accessed February 4, 2026, <https://www.beckhoff.com/en-us/products/automation/twincat/tcxxxx-twincat-3-base/tc1000.html>
7. ADS - Beckhoff Information System, accessed February 4, 2026, <https://infosys.beckhoff.com/content/1033/tcinfosys3/11291871243.html>

8. TF5200 | TwinCAT 3 CNC - Programming manual - download - Beckhoff, accessed February 4, 2026, https://download.beckhoff.com/download/document/automation/twincat3/TF5200_programming_manual_en.pdf
9. SUL - Esolang, accessed February 4, 2026, <https://esolangs.org/wiki/SUL>
10. Basics of Structured Text (ST) Programming | Examples & Applications - RealPars, accessed February 4, 2026, <https://www.realpars.com/blog/structured-text>
11. The Canonical Robot Command Language (CRCL) - PMC - NIH, accessed February 4, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC5436486/>
12. XML file describing the controls for a KUKA robot with a fixed base. - ResearchGate, accessed February 4, 2026, https://www.researchgate.net/figure/ML-file-describing-the-controls-for-a-KUKA-robot-with-a-fixed-base_fig7_281899019
13. Robotics and ISA 88 Batch Control Standard - Opportunities and Challenges Johnsson, Charlotta - LUCRIS, accessed February 4, 2026, <https://lucris.lub.lu.se/ws/files/6185913/7763021.pdf>
14. XML Exchange and PLCopen, accessed February 4, 2026, <https://www.plcopen.org/standards/xml-echange/>
15. PLCopen Motion Control Standard, accessed February 4, 2026, <https://www.plcopen.org/standards/motion-control/>
16. Increased Motion Control Programming Efficiency - International Society of Automation (ISA), accessed February 4, 2026, <https://www.isa.org/intech-home/2021/april-2021/features/increased-motion-control-programming-efficiency>
17. TF6421 - TwinCAT 3 | XML Server - download - Beckhoff, accessed February 4, 2026, https://download.beckhoff.com/download/document/automation/twincat3/TF6421_TC3_XML_Server_EN.pdf
18. Overview - Beckhoff Information System - English, accessed February 4, 2026, https://infosys.beckhoff.com/content/1033/tf6421_tc3_xml_server/187110539.html
19. Simulation and Prediction of Springback in Sheet Metal Bending Process Based on Embedded Control System - PMC - NIH, accessed February 4, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC11645057/>
20. Adaptive bending - The Fabricator, accessed February 4, 2026, <https://www.thefabricator.com/thefabricator/article/bending/adaptive-bending>
21. Cybersecure ISA-88 recipes and control with IEC 61131-3, accessed February 4, 2026, <https://www.isa.org/intech-home/2018/november-december/features/cybersecure-isa-88-recipes-and-control-with-iec-61>
22. What Does a Procedure Look Like? The ISA S88.02 Recipe Representation Format, accessed February 4, 2026, <https://www.yokogawa.com/us/library/resources/media-publications/what-does-a-procedure-look-like-the-isa-s8802-recipe-representation-format/>
23. Adaptive Control Design - MATLAB & Simulink - MathWorks, accessed February 4, 2026, <https://www.mathworks.com/help/slcontrol/adaptive-control-design.html>

24. Scalable and powerful: The open CNC solution for machine tools - Beckhoff, accessed February 4, 2026, https://www.beckhoff.com/media/downloads/information-media/beckhoff_machine_tools.pdf
25. "Interpreter" tab - Beckhoff Information System - English, accessed February 4, 2026, https://infosys.beckhoff.com/content/1033/tf5100_tc3_nc_i/3288335883.html
26. Writing CSV files - Beckhoff Information System - English, accessed February 4, 2026, https://infosys.beckhoff.com/content/1033/tf6420_tc3_database_server/5778527243.html
27. TwinCAT 3 Tutorial: Structured Text - Contact and Coil, accessed February 4, 2026, <https://www.contactandcoil.com/twincat-3-tutorial/structured-text/>
28. Beckhoff How to Open and Read a file in TWINCAT 3 C++ in a CycleUpdate?, accessed February 4, 2026, <https://stackoverflow.com/questions/51756724/beckhoff-how-to-open-and-read-a-file-in-twincat-3-c-in-a-cycleupdate>
29. Sequential Function Chart (SFC) Programming for Beginners - YouTube, accessed February 4, 2026, <https://www.youtube.com/watch?v=xCONAyL4Oow>
30. Maximize WPF 3D Performance - Microsoft Learn, accessed February 4, 2026, <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/maximize-wpf-3d-performance>
31. Optimizing Performance: Taking Advantage of Hardware - WPF | Microsoft Learn, accessed February 4, 2026, <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/optimizing-performance-taking-advantage-of-hardware>
32. How to Improve WPF Application Performance | SciChart, accessed February 4, 2026, <https://www.scichart.com/blog/how-to-improve-wpf-application-performance/>
33. Helix Toolkit is a collection of 3D components for .NET. - GitHub, accessed February 4, 2026, <https://github.com/helix-toolkit/helix-toolkit>
34. How to: Render on a Per Frame Interval Using CompositionTarget - WPF | Microsoft Learn, accessed February 4, 2026, <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/how-to-render-on-a-per-frame-interval-using-compositiontarget>
35. Why is Frame Rate in WPF Irregular and Not Limited To Monitor Refresh? - Stack Overflow, accessed February 4, 2026, <https://stackoverflow.com/questions/5812384/why-is-frame-rate-in-wpf-irregular-and-not-limited-to-monitor-refresh>
36. What is Sheet Metal Bending? : Our Guide | Dassault Systèmes, accessed February 4, 2026, <https://www.3ds.com/store/cad/sheet-metal-bends>
37. Is WPF 3D good alternate of DirectX and OpenGL for complex applications? [closed], accessed February 4, 2026, <https://stackoverflow.com/questions/8941383/is-wpf-3d-good-alternate-of-directx-and-opengl-for-complex-applications>

38. What is glTF: A Complete Guide - VividWorks, accessed February 4, 2026,
<https://www.vividworks.com/blog/what-is-gltf-a-complete-guide>
39. GLTF / GLB file format for 3D Assembly Instructions from CAD, accessed February 4, 2026,
<https://www.dyvixion.com/blog/gltf-glb-for-3d-assembly-instructions-from-cad>
40. Communication modes - Beckhoff Information System - English, accessed February 4, 2026,
https://infosys.beckhoff.com/content/1033/tf3710_tc3_interface_for_labview/9919635851.html
41. Use of ADS Notifications - Beckhoff Information System - German, accessed February 4, 2026,
<https://infosys.beckhoff.com/content/1031/tcadsnetref/7312578699.html>
42. CompositionTarget.Rendering event slows down when there are no UI updates · Issue #1908 · dotnet/wpf - GitHub, accessed February 4, 2026,
<https://github.com/dotnet/wpf/issues/1908>
43. Overcoming the limitations of adaptive control by means of logic-based switching - Daniel Liberzon - University of Illinois, accessed February 4, 2026,
<http://liberzon.csl.illinois.edu/research/ppsurvey.pdf>
44. Scripting API: RaycastHit.triangleIndex - Unity - Manual, accessed February 4, 2026,
<https://docs.unity3d.com/6000.3/Documentation/ScriptReference/RaycastHit-triangleIndex.html>
45. 3D Hit Testing in WPF - Stack Overflow, accessed February 4, 2026,
<https://stackoverflow.com/questions/6817106/3d-hit-testing-in-wpf>
46. Determine if ray hits an edge of a model in Unity 3D - Game Development Stack Exchange, accessed February 4, 2026,
<https://gamedev.stackexchange.com/questions/163620/determine-if-ray-hits-an-edge-of-a-model-in-unity-3d>