Vehicle Machine Language (VML)

White Paper Draft v0.1

Technical Working Group for Formal Automotive Systems

Open, vendor-neutral working draft

Natural language belongs to engineering.
Formal languages belong to runtime systems.

July 26, 2026

Status: initiative phase; not a final standard.

Contents

Abstract

Modern automotive systems increasingly combine perception, reasoning and planning through AI-based architectures, including foundation models and vision-language-action systems. These systems can support engineering, simulation and specification activities, but natural language itself is not a suitable runtime execution language for safety-critical vehicle behavior. It is ambiguous, context-dependent, difficult to certify, and unsuitable as the sole basis for deterministic execution.

VML is proposed as an open standard for formal behavioral representation in AI-driven automotive systems. It provides a formally defined intermediate representation between human-level requirements and runtime vehicle behavior. Foundation models may act as semantic compilers from requirements, scenarios and engineering intent into VML; however, runtime systems operate exclusively on VML artifacts with deterministic semantics, type checks, monitor semantics and verification interfaces.

The long-term transformation chain is:

$$\text{Natural Language} \longrightarrow \text{VML} \longrightarrow \text{Vehicle Behavior.}$$

The purpose of this white paper is to define the motivation, architectural role, design principles, initial language layers and standardization path for VML as an open, vendor-neutral foundation for transparent, verifiable and certifiable autonomous driving behavior.

1 Motivation

1.1 The problem: natural language at the edge of runtime

Natural language is excellent for communication between humans. It supports negotiation, underspecification, context and pragmatic interpretation. These properties are beneficial during requirements engineering, safety analysis and architecture discussions, but they become liabilities when natural language directly influences safety-critical runtime decisions.

A runtime decision system must support repeatable interpretation, testability, traceability and safety argumentation. A sentence such as “yield to pedestrians when necessary” may be intuitive to engineers, yet it leaves open the meaning of *pedestrian*, *yield*, *necessary*, the relevant time horizon, the applicable confidence threshold, and what fallback obligation applies when the primary obligation cannot be fulfilled. Such ambiguity is unacceptable as the final representation consumed by a vehicle controller.

1.2 The principle: separate human communication from machine execution

VML enforces a separation of concerns:

- Natural language is used at engineering time for expressing intent, requirements, scenarios and design rationale.
- Foundation models may translate or assist in translating that intent into formal specifications.
- Runtime systems consume only formal VML artifacts: typed, checked, monitorable and executable.

This makes modern AI useful without making opaque natural-language prompts part of the safety-critical runtime contract.

1.3 Why an open standard?

Future automotive AI should not depend on proprietary behavioral languages. The behavior representation that connects engineering, verification and runtime execution should be open, inspectable and implementable by multiple organizations. An open standard enables interoperability, independent verification, transparent certification, academic research, regulatory collaboration and long-term maintainability.

The goal of VML is not to standardize AI models, perception stacks or planners. The goal is to standardize the formal representation of behavioral intent that such systems generate, consume, verify or monitor.

2 Vision and Scope

2.1 Vision

VML is envisioned as the formal behavioral representation inside future AI-based driving systems. It provides a common semantic target for requirements, traffic rules, safety constraints, planning goals, deontic policies, probabilistic perception outputs and runtime monitor verdicts.

In the intended architecture, a foundation model is not a runtime authority. It is a semantic compiler that produces a formal program. The formal program is then parsed, type-checked, verified, monitored and executed through deterministic components.

2.2 In scope for the initial standardization effort

The initial scope includes:

- a core logical and temporal language for behavioral constraints;
- metric comparisons over vehicle and environment state;
- goal specifications for planners;
- deontic constructs for obligations, permissions and prohibitions;
- optional probabilistic constructs for uncertain perception outputs;
- optional ontology constructs for class membership and inheritance;
- optional utility constructs for lexicographic optimization;
- static semantics, dynamic semantics and runtime monitor outputs;
- reference execution pipeline guidance;
- conformance levels and conformance tests.

2.3 Out of scope for version 0.1

The following topics are intentionally out of scope for the first working draft:

- full denotational semantics for every probabilistic operator variant;
- a standardized ontology exchange format between organizations;
- complete certification argument templates;
- code generation for any specific vehicle platform;
- standardization of perception, planning or foundation-model implementations.

3 Reference Architecture

3.1 Pipeline

A reference VML pipeline is shown conceptually as:

Human Requirements
↓
Foundation Model as Semantic Compiler
↓
VML Specification
↓
Parser → AST → Type Checker → Formal Verification
↓
Runtime Monitor / Constraint Filter / Trajectory Optimizer
↓
Vehicle Controller

The decisive architectural constraint is that natural language does not directly control the vehicle. Natural language may influence the engineering process, but runtime behavior is mediated by formally checked VML specifications.

3.2 Reference execution pipeline

A minimal implementation pipeline is:

$$\text{Lexer} \rightarrow \text{Parser} \rightarrow \text{AST} \rightarrow \text{Type Checker} \rightarrow \text{Rule Engine / Monitor.}$$

A planner-integrated pipeline is:

Perception State → VML Evaluator → Constraint Filter → Trajectory Optimizer → Controller.

3.3 Online monitor loop

```
for each incoming state S_k:
  update atom table and numeric store
  evaluate guards and activate applicable deontic rules
  progress temporal monitors
  evaluate hard constraints and record violations
  update goals and optimization context
  publish verdict/event stream
```

4 System Model

4.1 Execution trace

A run is a discrete state trace

$$\sigma = S_0, S_1, S_2, \ldots$$

with sample period $\Delta t > 0$, fixed globally or piecewise fixed by implementation configuration.

4.2 State shape

Each state S_k may contain:

- Boolean atoms, for example `Collision = FALSE`;
- numeric variables, for example `Ego.Velocity = 7.3`;
- optional probability fields, for example `P(IsPedestrian(Obj1)) = 0.92`;
- optional symbolic class memberships, for example `Obj1 ISA Pedestrian`.

A VML implementation must define how perception, localization, prediction, map and planner state are mapped into this canonical state representation.

5 Layered Language Model

VML is structured as a composition of layers. Each layer addresses a fundamental aspect of autonomous behavior specification:

$$\text{VML} = L \oplus T \oplus M \oplus P \oplus G \oplus D \oplus O \oplus U.$$

Layers are integrated but can be implemented incrementally.

Layer	Name	Guiding question
L	Logical	What facts hold? What exists and what is true?
T	Temporal	When must it happen? When must it hold?
M	Metric	How much? Distances, velocities, thresholds and numeric constraints.
P	Probabilistic	How certain is perception or prediction?
G	Goal	What outcomes are desired?
D	Deontic	What is obligatory, permitted or forbidden?
O	Ontology	What classes and inheritance relationships apply?
U	Utility	How are soft objectives ranked?

5.1 Core and extended profiles

VML defines two initial conformance profiles:

Core profile. Required for 0.1 conformance. Includes logic, temporal operators, metric comparisons, goals, obligations and monitor semantics.

Extended profile. Optional for 0.1. Includes probabilistic expressions, ontology declarations, utility optimization objectives, contrary-to-duty deontics and assumptions.

6 Core Semantics

6.1 Boolean and temporal semantics

Let $\sigma, k \vDash \varphi$ mean that formula φ holds at state index k in trace σ. Core temporal semantics are:

$$\begin{aligned}
\sigma, k \vDash \neg\varphi &\Leftrightarrow \neg(\sigma, k \vDash \varphi),\\
\sigma, k \vDash \varphi \wedge \psi &\Leftrightarrow (\sigma, k \vDash \varphi) \wedge (\sigma, k \vDash \psi),\\
\sigma, k \vDash \varphi \vee \psi &\Leftrightarrow (\sigma, k \vDash \varphi) \vee (\sigma, k \vDash \psi),\\
\sigma, k \vDash \varphi \Rightarrow \psi &\Leftrightarrow \neg(\sigma, k \vDash \varphi) \vee (\sigma, k \vDash \psi),\\
\sigma, k \vDash \textsc{Next}\ \varphi &\Leftrightarrow \sigma, k+1 \vDash \varphi,\\
\sigma, k \vDash \textsc{Always}\ \varphi &\Leftrightarrow \forall j \geq k : \sigma, j \vDash \varphi,\\
\sigma, k \vDash \textsc{Eventually}\ \varphi &\Leftrightarrow \exists j \geq k : \sigma, j \vDash \varphi,\\
\sigma, k \vDash \varphi\ \textsc{Until}\ \psi &\Leftrightarrow \exists j \geq k : \sigma, j \vDash \psi \wedge \forall i \in [k, j) : \sigma, i \vDash \varphi.
\end{aligned}$$

6.2 Deontic monitor semantics

For monitor semantics:

- `OBLIGATORY e` produces a violation when e is false in a state where it applies;
- `PROHIBITED e` produces a violation when e is true in a state where it applies;
- `PERMITTED e` produces no direct violation and is used as policy metadata or a planner tie-break signal;
- a `WHEN c` guard limits applicability to states where c is true.

6.3 Contrary-to-duty obligations

Safety-critical behavior often contains sub-ideal cases. A primary obligation may be impossible without violating another obligation. VML therefore supports contrary-to-duty logic:

$$\textsc{Obligatory}\ \alpha\ \textsc{Otherwise Obligatory}\ \beta.$$

If α is violated or physically impossible in an applicable state, then β becomes the secondary obligation. This is the formal basis for minimal-risk maneuvers and fallback behavior.

7 Concrete Syntax Draft

The following grammar sketches the first working draft syntax. It is intended as a normative starting point for parser behavior and conformance testing.

```
program      ::= declaration* statement*

declaration  ::= class_decl | binding_decl
class_decl   ::= "CLASS" IDENT "ISA" IDENT ("HAS" "{" attr_list "}")? ";"
attr_list    ::= attr ("," attr)*
attr         ::= IDENT ":" TYPE
binding_decl ::= IDENT "=" literal ";"

statement    ::= goal_stmt
               | hard_rule
```

```
                | deontic_stmt
                | require_stmt
                | optimize_stmt
                | assume_stmt

goal_stmt     ::= "GOAL" expr ";"

hard_rule     ::= "ALWAYS" expr ";"
                | "EVENTUALLY" expr ";"
                | "NEXT" expr ";"
                | expr "UNTIL" expr ";"
                | expr ";"

require_stmt ::= "REQUIRE" "(" expr ")" "IMPLIES" statement

optimize_stmt ::= "OPTIMIZE" "PRIORITY" "(" INT ")"
                   ("MINIMIZE" | "MAXIMIZE") expr ";"

assume_stmt   ::= "ASSUME" expr ";"

deontic_stmt ::= deontic_op expr ("WHEN" expr)?
                   ("OTHERWISE" deontic_stmt)? ";"
deontic_op    ::= "OBLIGATORY" | "PERMITTED" | "PROHIBITED"

expr          ::= impl_expr
impl_expr     ::= or_expr ("IMPLIES" or_expr)*
or_expr       ::= and_expr ("OR" and_expr)*
and_expr      ::= unary_expr ("AND" unary_expr)*
unary_expr    ::= "NOT" unary_expr | temporal_expr

temporal_expr ::= "ALWAYS" unary_expr
                 | "EVENTUALLY" unary_expr
                 | "NEXT" unary_expr
                 | primary_expr ("UNTIL" primary_expr)?

primary_expr ::= "(" expr ")"
                | predicate
                | comparison
                | prob_expr
                | isa_expr

predicate     ::= IDENT ("(" arg_list ")")?
arg_list      ::= expr ("," expr)*
comparison    ::= value comparator value
comparator    ::= "<" | "<=" | "=" | ">=" | ">"
value         ::= IDENT | NUMBER | STRING | BOOLEAN
isa_expr      ::= IDENT "ISA" IDENT

prob_expr     ::= "P" "(" expr ")"
                | "EXPECTED" "(" expr ")"
                | "CDF" "(" expr ")"
```

```
              | "EXPECTED_RISK" "(" arg_list ")"

literal      ::= NUMBER | STRING | BOOLEAN
TYPE         ::= "BOOLEAN" | "REAL" | "INT" | "STRING"
IDENT        ::= /[A-Za-z_][A-Za-z0-9_.]*/
NUMBER       ::= /-?[0-9]+(\.[0-9]+)?/
STRING       ::= /"([^"\\]|\\.)*"/
BOOLEAN      ::= "TRUE" | "FALSE"
INT          ::= /[0-9]+/
```

8 Static Semantics

An implementation must reject programs that violate static checks. Core checks include:

1. every referenced identifier has a declaration source, such as a state schema, binding or class member;
2. Boolean operators consume Boolean subexpressions;
3. numeric comparators consume numeric terms;
4. `UNTIL` has Boolean left and right operands;
5. `GOAL` expressions are Boolean;
6. deontic content for `OBLIGATORY`, `PROHIBITED` and `PERMITTED` is Boolean.

Optional checks include:

1. `P(...)` and `CDF(...)` evaluate to values in $[0, 1]$;
2. `PRIORITY(n)` uses positive integer levels;
3. `ISA` relationships are acyclic;
4. probabilistic estimator sources and update frequencies are declared by implementation metadata.

9 Layer-by-Layer Guidance

9.1 Logical layer

Purpose: define vocabulary, facts and relations.

```
PedestrianInCrosswalk;
InLane(Ego, Lane_2);
```

Implementations should maintain a symbol table for atoms and predicates, map sensor and planner outputs to canonical atom updates, and use stable naming conventions such as `Ego.Velocity` or `Obj12.IsPedestrian`.

9.2 Temporal layer

Purpose: encode persistence, liveness and ordered behavior.

```
ALWAYS (NOT Collision);
Braking UNTIL CrossingClear;
```

Temporal formulas may be compiled to monitor automata or progression states. Online monitoring requires per-formula runtime memory for open obligations and pending eventualities.

9.3 Metric layer

Purpose: encode numeric safety envelopes and dynamic limits.

```
ALWAYS (DistanceToPedestrian > 2.0);
ALWAYS (Ego.LateralAcceleration < 2.5);
```

Implementations should normalize units at ingestion and evaluate numeric comparisons after unit conversion and signal validity checks.

9.4 Probabilistic layer

Purpose: preserve uncertainty instead of collapsing uncertain perception into premature hard thresholds.

```
REQUIRE (P(IsPedestrian(Obj1)) > 0.95)
IMPLIES OBLIGATORY (Ego.Velocity = 0);
```

Perception systems output probabilistic classifications and state estimates. VML bridges the perception-planning gap by representing world entities as stochastic states and by introducing operators over probability density functions, cumulative distribution functions and expectations.

9.5 Goal layer

Purpose: express desired outcomes pursued while respecting hard constraints.

```
GOAL At(Ego, Destination);
```

Goals must not override hard safety violations. Implementations should maintain explicit goal status such as inactive, active, reached or blocked.

9.6 Deontic layer

Purpose: encode obligations, permissions, prohibitions and fallback duties.

```
OBLIGATORY YieldToPedestrian WHEN PedestrianInCrosswalk;
PROHIBITED CrossSolidLine;
```

A monitor should emit structured violation records including timestamp, rule identifier and witness signals.

9.7 Ontology layer

Purpose: reduce duplication by class inheritance.

```
CLASS VulnerableRoadUser ISA DynamicObstacle;
CLASS Pedestrian ISA VulnerableRoadUser;
OBLIGATORY YieldTo(target) WHEN (target ISA VulnerableRoadUser);
```

If `Pedestrian ISA VulnerableRoadUser` is declared in the ontology, any detected pedestrian inherits rules written for vulnerable road users. This simplifies verification and reduces semantic compile size.

9.8 Utility layer

Purpose: represent soft objectives under strict priority ordering.

```
OPTIMIZE PRIORITY(1) MINIMIZE EXPECTED_RISK(Collision);
OPTIMIZE PRIORITY(2) MINIMIZE Ego.LateralJerk + Ego.LongitudinalJerk;
OPTIMIZE PRIORITY(3) MAXIMIZE Ego.Velocity;
```

A higher-priority objective must not be compromised for a lower-priority objective. This supports multi-objective trajectory generation without hiding hard safety constraints inside a monolithic cost function.

10 Probabilistic and Statistical Extensions

10.1 Probabilistic entity attributes

Every dynamic entity e may have an associated probabilistic state vector. Example attributes include:

- $P(e \text{ ISA } C)$: categorical probability that entity e belongs to class C;
- $e.X \sim \mathcal{N}(\mu, \Sigma)$: spatial state vector represented as a distribution;
- confidence, covariance or estimator metadata suitable for downstream risk computations.

10.2 Expected risk

For safety-critical probabilistic terms, VML may define an operator such as:

$$R(\text{Collision}) = \sum_{c \in \text{Classes}} P(e \text{ ISA } c) \int_{x_e} p(x_{ego}) p(x_e \mid c)\, dx \times \text{Severity}(c).$$

This supports rules over expected harm rather than rules over brittle binary object classifications.

```
-- Stop rule based on collision probability and entity class severity
ALWAYS REQUIRE (EXPECTED_RISK(Collision, Ego, Obstacle_1) > 10^-5)
IMPLIES
  OBLIGATORY (Ego.Velocity = 0 UNTIL
              EXPECTED_RISK(Collision, Ego, Obstacle_1) <= 10^-6);
```

11 Spatial-Topological Logic

Autonomous driving rules often become brittle when expressed only in absolute coordinates. VML should therefore support qualitative spatial relations, inspired by region connection calculi, over two-dimensional regions such as bounding boxes, lanes, safety envelopes and crosswalk areas.

Example predicates include:

- `Spatial.DC(A,B)`: disconnected;
- `Spatial.EC(A,B)`: externally connected;
- `Spatial.PO(A,B)`: partial overlap;
- `Spatial.TPP(A,B)`: tangential proper part.

```
-- Formalizing safe keeping using topological relations
ALWAYS PROHIBITED (Spatial.PO(Ego.BoundingBox, Vehicle_1.SafetyEnvelope));
```

12 Multi-Agent Assumptions

Safety guarantees depend on explicit assumptions about other agents. Overly optimistic assumptions such as "other drivers always obey stop signs" are unsafe. Overly pessimistic assumptions such as "other drivers may accelerate into ego at any second" lead to freezing behavior. VML therefore includes conditional assumptions to define game-theoretic boundaries of cooperation.

```
-- Assume cooperative play under bounded rationality
ASSUME ALWAYS (EXPECTED_RISK(Collision, OtherVehicle, Ego) < 10^-2)
IMPLIES
  ALWAYS OBLIGATORY (MaintainSmoothTrajectory(Ego));
```

13 Unified Use Case: Unprotected Left Turn

The following example illustrates how ontology, probabilistic risk, deontic fallback and utility objectives can be combined.

```
-- Unprotected Left Turn VML Specification (illustrative draft)

-- 1. Ontological mapping
CLASS OncomingVehicle ISA Vehicle;
CLASS CrosswalkPedestrian ISA Pedestrian;

-- 2. Utility layer optimization goals
OPTIMIZE PRIORITY(1) MINIMIZE EXPECTED_RISK(Collision);
OPTIMIZE PRIORITY(2) MINIMIZE Ego.LateralJerk;
OPTIMIZE PRIORITY(3) MAXIMIZE Ego.Velocity;

-- 3. Behavioral rules
-- Rule A: hard obligation to stop for pedestrian if risk is high
ALWAYS REQUIRE (EXPECTED_RISK(Collision, Ego, target) > 10^-5
                AND target ISA Pedestrian)
IMPLIES
  OBLIGATORY (Ego.Velocity = 0.0)
  OTHERWISE OBLIGATORY (
    Ego.Maneuver = SafeBrakingToShoulder
  );

-- Rule B: execute unprotected left turn only when gap is statistically safe
ALWAYS OBLIGATORY (
  Ego.Maneuver = ExecuteLeftTurn
)
WHEN (
  EXPECTED_RISK(Collision, Ego, OncomingVehicle) < 10^-6 AND
  CDF(OncomingVehicle.Velocity > 25.0) < 0.02
)
OTHERWISE OBLIGATORY (
  Ego.Velocity = 0.0 AND Ego.Maneuver = YieldAtIntersection
);

-- Rule C: respect spatial safety boundaries topologically
ALWAYS PROHIBITED (
```

```
  Spatial.PO(Ego.BoundingBox, Crosswalk.Boundary)
)
WHEN (
  EXPECTED_RISK(Collision, Ego, target) > 10^-6 AND
  target ISA VulnerableRoadUser
);
```

14 Natural Language to VML Translation Workflow

A recommended engineering workflow is:

1. parse the requirement sentence into actor, condition, action and timing;
2. map domain terms to canonical ontology symbols;
3. distinguish hard constraints from goals and soft objectives;
4. introduce numeric thresholds and units explicitly;
5. add fallback deontic obligations for infeasible primary duties;
6. run type checks and simulation traces;
7. revise the VML program until monitor verdicts are stable and reviewable.

This workflow is compatible with human authoring, AI-assisted drafting, static analysis and formal review. A foundation model may generate candidate VML programs, but conformance depends on deterministic parsing, type checking, test execution and reviewable semantics.

15 Normative Runtime Outputs

A conforming runtime monitor should expose at least:

- `rule_id`: identifier of the evaluated rule;
- timestamp: state index and wall time if available;
- status in {`satisfied`, `violated`, `pending`};
- evidence: minimal set of signal values used in the decision;
- optional confidence or estimator metadata for probabilistic decisions;
- optional remediation or fallback obligation identifiers.

16 Verification and Certification Interfaces

VML is designed to support several complementary verification methods:

- syntactic conformance tests for parsers;
- type and consistency checks for specifications;
- model checking for finite abstractions;
- theorem-proving hooks for semantic fragments;

- runtime monitoring over observed traces;
- scenario-based simulation against expected verdict streams;
- regression tests for natural-language-to-VML compiler output.

A key design goal is that authorities, auditors and development partners can inspect behavioral rules without requiring disclosure of proprietary perception network weights, generative model parameters or planner internals.

17 Conformance Strategy

17.1 Profile-based conformance

Version 0.1 should define profile-based conformance:

Core parser conformance. The implementation accepts and rejects programs according to the normative grammar.

Core monitor conformance. The implementation produces required verdicts for Boolean, temporal, metric, goal and deontic constructs.

Extended feature conformance. The implementation documents support for probabilistic, ontology and utility layers.

Trace conformance. The implementation produces reproducible verdict streams over standardized input traces.

17.2 Reference test artifacts

The working group should maintain:

- grammar acceptance and rejection suites;
- typed example programs;
- canonical state traces;
- expected monitor verdict streams;
- negative examples for ambiguity, type errors and unsupported constructs;
- implementation reports for independent tools.

18 Open Standard Development Model

18.1 Vendor-neutral collaboration

VML should be developed by a technical working group open to automotive OEMs, tier-1 suppliers, semiconductor vendors, software companies, universities, research institutes, standardization organizations, government agencies and independent researchers.

18.2 Governance principles

The open standard process should follow these principles:

- public working drafts and public issue tracking where possible;
- clear distinction between normative and informative text;
- reference implementations under permissive licensing where feasible;
- conformance tests published alongside the specification;
- compatibility policy for future versions;
- transparent intellectual-property and contribution rules;
- review by safety, formal-methods and automotive-domain experts.

19 Roadmap

A pragmatic roadmap is:

1. stabilize the motivation, architecture and terminology;
2. publish this white paper as the entry document for the initiative;
3. publish VML language specification draft v0.1;
4. assemble a minimal reference parser and monitor;
5. define conformance traces and expected verdicts;
6. refine probabilistic, deontic, ontology and utility extensions;
7. engage external contributors and establish formal working mode;
8. prepare a public draft standard candidate.

20 Conclusion

VML proposes a practical formal foundation for AI-driven vehicle behavior: natural language at design time, formal language at runtime. It allows foundation models to support engineering as semantic compilers while preserving deterministic runtime semantics, verification, monitorability and certification traceability.

The central claim is simple:

$$\text{Natural Language} \neq \text{Runtime Language}.$$

By standardizing the formal representation of behavioral intent rather than the AI models themselves, VML can become a common interface between requirements engineering, AI-assisted specification, simulation, verification, planning and deployment. As an open standard, it can help the automotive ecosystem build transparent and certifiable AI-driven systems without locking safety-critical behavior into proprietary or ambiguous representations.

A Glossary

Behavioral intent The intended vehicle behavior expressed as rules, goals, constraints and policies.

Deontic rule A rule expressing obligation, permission or prohibition.

Foundation model A large AI model used here as an engineering-time semantic compiler, not as a runtime authority.

Intermediate representation A formal language between human requirements and executable vehicle behavior.

Monitor verdict A runtime output indicating whether a rule is satisfied, violated or pending over a trace.

Open standard A specification developed through an open, vendor-neutral process with implementable conformance criteria.

B Minimal Website Link Integration

For the website draft, the document can be linked as:

```
<a href="docs/VML_whitepaper_v0_1.pdf" class="doc-link">PDF</a>
```

A matching source file can be maintained as:

```
docs/VML_whitepaper_v0_1.tex
```