

STIL Name Spaces

From STIL quick reference guide (syntax summary)

STIL Block	Type of Name	Domain Restrictions
Variables	Variables domain names	Supports a single unnamed block and domain named blocks. Domain names shall be unique across all Variables blocks.
Variables, Signals, SignalGroups, Spec	Signal names SignalGroup names SignalVariable names Spec variable names Integer names IntegerConstant names WFCConstant names	Names are present in this name space are dependent on the domain reference statements in the Pattern-Burst (SignalGroups domains, Variables domains) and the PatternExec (Category domains). It is an error for defined Variable or Constant names to conflict with Signals, SignalGroups, or Spec variable names accessible in the same context.

From dot0

6.16 STIL name spaces and name resolution

Information defined in a block with a domain name requires a reference to that domain name to use the information in that block. Information defined in a block without a domain name is available to be used without an explicit reference; this information is considered “global” after it has been defined. All information in a named domain becomes available when that name is referenced. Because most STIL domain blocks contain only data definitions, there is no method to access a subset of domain information, except for Timing information, which may have several levels of hierarchy and scoping mechanisms. (See 6.11 for more information.) If a subset of information is desired, then the information may be partitioned into separate named blocks.

There are three environments for name spaces in STIL.

- The first supports both unnamed (global) definitions and domain named definitions. This environment is used for signals and groups (although the name space of signals is global only, it is combined with group names that support this environment), procedures, macros, and timing names.
- **The second name space environment is global only, used for spec variables (and the signal name subset of signal and group blocks).**
- The third environment is where the domain name itself serves as the name. This environment is used by the selector, patternburst, and pattern names.

The first environment requires mechanisms to resolve potential conflicts between unnamed (global) information and names declared under a domain name. These mechanisms are as follows:

- A name defined in a SignalGroups, Timing, Procedures, and MacroDefs block with a domain name for that block shall override any identical names of the same type defined from a block with no domain name, when that domain name is referenced.
- It is an error to refer to a name defined in two different (domain named) blocks, even if both definitions for the name are the same.

- Specifically for the SignalGroups information: it is an error to define a group name in a SignalGroups block with no domain name, with the same name as a name defined in a Signals block. Signal names may be overridden only from SignalGroups with domain names.
- For spec variables, it is an error to redefine a previously defined category for a spec variable.
- For selector, patternburst, and pattern name spaces: it is an error to redefine a previously defined name.
- It is an error to redefine a previously defined signal name, although a domain named SignalGroup may override a named signal. In the case that a domained SignalGroup changes the definition of a signal name, that new definition shall be reflected in all signal groups that use that name, including groups defined in the global SignalGroup block, when that domained SignalGroup is referenced.

Table 6—STIL name spaces

STIL Block	Type of Name	Domain Restrictions
Signals	Signal and SignalGroup names ¹	Any signal defined in the Signal block is global.
SignalGroups	Signal and SignalGroup names ²	Supports a single unnamed global block and domain name (restricted) blocks.
ScanStructures	Scan names and scanchain names	A single ScanStructures block is optionally named. Multiple ScanStructures blocks shall be uniquely named. ScanChain names shall be unique inside a ScanStructures block.
Procedures	Procedure names	Supports a single unnamed global block and domain name (restricted) blocks.
MacroDefs	Macro names	Supports a single unnamed global block and domain name (restricted) blocks.
Timing	Waveform table names ³	Supports a single unnamed global block and domain name (restricted) blocks.
Timing	Event_labels	Event_label names shall be unique with respect to Spec variables, but may be multiply declared if scoped within Timing information.
Spec	Spec names	There may be multiple Spec blocks present. The names of all Spec blocks shall be unique.
Spec	Spec variables, spec categories	Any Spec variable or Spec category defined in any Spec block is global, irrespective of the presence of an explicit domain name on the Spec block
Selector	Selector names	Each Selector block defines an entity. The domain name is required.
PatternBurst	Pattern/PatternBurst names	Each PatternBurst or Pattern block defines an entity. The domain name is required, and the name space of Pattern and PatternBurst blocks is shared.
Pattern	Pattern/PatternBurst names	
PatternExec	PatternExec names	PatternExec PatternExec names Supports a single unnamed global block and domain name blocks.

¹ The Signals block defines only signals, but the name space is shared for both groups.

² The SignalGroups block defines only groups, but the name space is shared for both groups.

³ Also contains hierarchical data items, discussed subsequently.

From dot1

9. Variables block

This clause defines variables and named constants. If the block is unnamed, then all definitions shall be globally available to all other blocks in the STIL file/stream. If the block has a name, then that name must be referenced by the Pattern, PatternBurst, or Timing block for the variables to be available. **Variables and**

constants that are defined in the global Variables block shall not be overridden in a named Variables block.

This block should be considered in light of the Spec block that also can define constants. The purpose and usage of these two blocks is sufficiently different to warrant their separation. The Spec block is intended to contain parametric data used in specifying device/test characteristics. The Spec block has special handling defined for Category, Min/Typ/Max/Meas, and Selector, which are to facilitate a flexible definition of these test parameters.

The Variables block is used to contain control variables and constants. Typical uses for these constants and variables is to control flow of execution, to provide 'aliasing' of values to meaningful names with constants, and to provide run-time parameters.

Although complex expressions may be supported by an ATE system, in general, most ATE systems will be able to support only a limited subset of the design capabilities defined herein. The full capabilities are intended for use by simulation applications or pattern translation tools. See Annex O for more information.

9.3 Variables scoping

Care should be taken when defining Variables blocks as to whether the variables and constants are to be shared or unique to a pattern. Shared (global) variables are useful for passing data, whereas nonshared (local) variables provide independent operation. The scope of a variable or constant is determined by (1) whether the Variables block is named or unnamed and (2) how a named Variables block is referenced.

The example below shows the various usage models. This example uses integer-variables, but the same scoping rules apply to signal-variables, integer-constants, and WFC-constants.

Variables and constants declared in an unnamed Variables block are global in scope and may be accessed from any pattern context unless that variable is hidden by a declaration of the same name from a named Variables block referenced from a PatternBurst context. Global variables are set to the InitialValue at the start of a PatternExec and maintain the last-assigned value throughout the execution of the PatternExec.

Variables declared in named Variable blocks are local to the Pattern or PatternBurst context where that named Variable block is referenced. Local variables are assigned their InitialValue when the first Pattern that contains that Variables block reference starts to execute, and the value of that variable only persists while the Pattern or PatternBurst context that contains that Variables reference is executing. Each reference of a Variables block in a PatternBurst context defines separate and unique instances of local variables for that context.

STIL.0, Subclause 6.16 defines the rules for name resolution of signals and signal-groups across named and unnamed domains. Because variables exist in the same name space as signals and signal-groups, the same rules apply to the naming of variables. This is illustrated in the following example, where there is a signal named FOO and a signal-variable named FOO that overrides the global signal name.

STIL dot1 blocks which can contain Variables <var_domain> statements.

PatternBurst

PatList

PatSet

ParallelPatSet

Timing

Environment

```
PatternBurst PAT_BURST_NAME {
  ( Variables VARIABLES_DOMAIN; )*
  ( F(ixed) { (cyclized-data)* } )
  ( PatList {
    ( PAT_NAME_OR_BURST_NAME {
      ( Variables VARIABLES_DOMAIN; )*
      ( If boolean_expr ; )
      ( While boolean_expr ; )
    })* // end pat_name_or_burst_name
  })* // end PatList
  ( PatSet {
    ( PAT_NAME_OR_BURST_NAME ; )*
    ( PAT_NAME_OR_BURST_NAME {
      ( Variables VARIABLES_DOMAIN; )*
    })* // end pat_name_or_burst_name
  })* // end PatSet
  ( ParallelPatList ( SyncStart | Independent | LockStep ) {
    ( PAT_NAME_OR_BURST_NAME ; )*
    ( PAT_NAME_OR_BURST_NAME {
      ( Variables VARIABLES_DOMAIN; )*
      ( Extend; )
      ( If boolean_expr ; )
      ( While boolean_expr ; )
    })* // end pat_name_or_burst_name
  })* // end ParallelPatList
})* // end PatternBurst

Timing TIMING_NAME {
  ( Variables VARIABLES_DOMAIN; )*
  ...
  ...
}

Environment (ENV_NAME) {
  ( NameMaps (MAP_NAME) {
    ( Signals { (SIG_NAME "MAP_STRING";)* } )*
    ( SignalGroups (DOMAIN_NAME) { (GROUP_NAME "MAP_STRING";)* } )*
    ( Variables { (VAR_NAME "MAP_STRING";)* } )*
    ( AllNames { (ANY_NAME "MAP_STRING";)* } )*
  })* // end NameMaps
} // end Environment
```