Chapter

53 *The SDL Validator*

This chapter is a reference to the validator user interface and a reference for the terminology used in validation.

For a guide to how to use the validator, see chapter 54, *Validating a System*.

The Validator Monitor

A validator generated by the SDL suite is similar to a simulator in that it contains an interactive monitor system. The validator monitor looks and behaves very similar to the simulator monitor; the only real difference is the set of available commands.

Monitor User Interfaces

Two different user interfaces are provided for the validator monitor, a textual and a graphical.

The textual interface only allows commands to be entered from the keyboard, in the same way as for the simulator monitor.

The graphical interface still allows commands to be entered from the keyboard in the same way, but also provides buttons, menus and dialogs for easy access to commands and other features.

The textual interface is invoked by executing the generated validator directly from the operating system prompt. This is called running a validator in *textual mode*. When started, the validator responds with the following text:

```
Welcome to the SDL VALIDATOR

Command :
```

Another prompt may appear if the SDL system contains external synonyms. For more information, see "Supplying Values of External Synonyms" on page 2326 in chapter 54, *Validating a System*.

> **Note:**
>
> Before a validator can be run in textual mode **on UNIX**, a command file must be executed from the operating system prompt. The file is called `telelogic.sou` or `telelogic.profile` and is located in the binary directory that is included in the user's path.
> For csh-compatible shells: **`source <bin dir>/telelogic.sou`**
> For sh-compatible shells: **`. <bin dir>/telelogic.profile`**

The graphical interface, known as the *Validator UI*, runs in a separate window. It is started from the Organizer by selecting *SDL > Validator UI* from the *Tools* menu. The graphical interface is basically the same as for the simulator monitor. However, some special windows and

menus have been added; see "Graphical User Interface" on page 2281 for more information.

Activating the Monitor

Commands can be issued to the interactive monitor system when it becomes active. The validator's monitor system becomes active:

- When the validator is started.
- When the last command was Next or Random-Down and the transitions initiated by this command have completed.
- When the last command was Bit-State-Exploration, Verify-MSC or Exhaustive-Exploration and the behavior tree down to the defined search depth has been fully explored.
- When the last command was Random-Walk and the defined number of repetitions have been executed.
- Immediately after a report with the report action Abort has been generated during automatic state space exploration.
- In the Validator UI, when the *Break* button is clicked in the *Explore* button module; in textual mode, when `<Return>` is pressed during an automatic state space exploration.

Note:

No other characters may be typed before `<Return>` is pressed.

Monitor Commands

This section provides an alphabetical listing of all available commands in the validator monitor. The syntax of commands and data type literals in the validator monitor is the same as for the simulator monitor. Commands may be abbreviated, as well as most command parameters. All input to the monitor is case insensitive. For more information, see "Syntax of Monitor Commands" on page 2064 and "Input and Output of Data Types" on page 2069 in chapter 50, *The SDL Simulator*.

? (Interactive Context Sensitive Help)

Parameters:

```
(None)
```

The monitor will respond to a '?' (question mark) by giving a list of all allowed values at the current position, or by a type name, when it is not suitable to enumerate the values. After the presentation of the list, the input can be continued.

? (Command Execution)

Parameters:

```
<Command name>
```

Executes the monitor command given as a parameter. This is a convenience function for the Validator UI. Entering '?' as parameter gives a list of all possible command names.

Assign-Value

Parameters:

```
[ '(' <PId value> ')' ]
<Variable name> <Optional component selection>
<New value>
```

The new value is assigned to the specified variable in the process instance, service instance, or procedure given by the current scope. Sender, Offspring, and Parent may also be changed in this way, but their names may not be abbreviated.

After the command is given, the root of the behavior tree is set to the current system state.

It is, in a similar way as for the command Examine-Variable, possible to handle components in structured variables (struct, strings, array, and Ref) by appending the struct component name or a valid array index be-

fore the value to be assigned. Nested structs and arrays can be handled by entering a list of index values and struct component names.

If a PId is given within parenthesis, the scope is temporarily changed to this process instance instead.

Bit-State-Exploration

Parameters:

`(None)`

Starts an automatic state space exploration from the current system state using the bit state space algorithm. When the exploration is started, the following information is printed:

- Search depth: The maximum search depth of the exploration. This can be set with the command Define-Bit-State-Depth.
- Hash table size: The size of the hash table used during bit state exploration. This can be set with the command Define-Bit-State-Hash-Table-Size.

The exploration will continue until either the complete state space to the defined depth is explored, `<Return>` is pressed from the command prompt, or the *Break* button is pressed in the graphical user interface. The system is then returned to the state it was in before the exploration was started.

If a bit state exploration already has been started, but has been stopped, this command asks if the exploration should continue from where it was stopped, or restart from the beginning again.

A status message is printed for every 20,000 transitions that are executed.

When the exploration is finished or stopped, the Report Viewer is by default opened (this can be changed with the command Define-Report-Viewer-Autopopup). The following statistics are also printed:

- No of reports: The number of reported situations that may be listed with the List-Reports command.
- Generated states: The total number of system states generated.
- Truncated paths: The number of times the exploration reached the maximum depth, causing the current execution path to be truncated.

- Unique system states: The number of generated system states that are not duplicated anywhere in the behavior tree.
- Size of hash table: The size of the hash table in bits and bytes.
- No of bits set in hash table: The number of bits actually used to represent the generated state space.
- Collision risk: The risk, in percent, of a collision occurring in the hash table for two different system states. This would cause an incorrect truncation of an execution path in a newly generated state.
- Max depth: The maximum number of levels in the behavior tree that have been reached during the exploration.
- Current depth: The level of the behavior tree reached at the moment when the exploration was stopped. If it is -1, the exploration was completed, i.e., the complete behavior tree down to the specified depth was explored. If it is > 0, the exploration may be continued from this level by issuing the command again.
- Min state size: The smallest number of bytes used to store a system state when computing hash values.
- Max state size: The largest number of bytes used to store a system state when computing hash values.
- Symbol coverage: The percentage of the symbols in the process graphs that have been executed at least once.

Bottom

Parameters:

```
(None)
```

Go down in the behavior tree to the end of the current path.

Cd

Parameters:

```
<Directory>
```

Change the current working directory to the specified directory.

Channel-Disable

Parameters:

```
<Channel> | '-'
```

Disables all test values defined for all signals using the given channel. '-' means *all channels*. Use Channel-Enable to start using them again. See also Signal-Disable. Test values for a signal are only used if both the signal and the channel that transports the signal are enabled. By default, all signals and channels are enabled.

Channel-Enable

Parameters:

```
<Channel> | '-'
```

Enables all test values defined for all signals using the given channel. '-' means *all channels*. See also Signal-Enable. Test values for a signal are only used if both the signal and the channel that transports the signal are enabled. By default, all signals and channels are enabled.

Clear-Autolink-Configuration

Parameters:

```
<Directory>
```

Clears the current Autolink configuration.

Clear-Constraint

Parameters:

```
<Constraint name> | '-'
```

Removes the constraint specified by the parameter. If '-' is given instead of a constraint name, all constraints are cleared.

Note: Only constraints that are not used by any generated test case can be cleared. Constraints used by one or more test cases are cleared automatically if the corresponding test cases are cleared.

Clear-Coverage-Table

Parameters:

```
(None)
```

Clears the test coverage information.

Clear-Generated-Test-Case

Parameters:

```
<Test case name> | '-'
```

Removes the generated test case specified by the parameter from memory. However, the corresponding MSC test case on disk is preserved. If '-' is given instead of a test case name, all generated test cases are

cleared. Alternatively, several test cases may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '*', '?', and '[...]' and is evaluated just as in a UNIX command shell.

Clear-Instance-Conversion

Parameters:

(None)

Clears all defined instance conversions. See also the command Define-Instance-Conversion.

Note that this command will not change an already loaded MSC.

Clear-MSC

Parameters:

(None)

Clears the currently loaded MSC and sets the current state to the current root of the behavior tree.

Clear-MSC-Test-Case

Parameters:

<File name> | '-'

Removes the MSC test case specified by the parameter from disk. If '-' is given instead of a file name, all test cases are cleared. Alternatively, several test cases may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '*', '?', and '[...]' and is evaluated just as in a UNIX command shell.

Clear-MSC-Test-Step

Parameters:

<File name> | '-'

Removes the MSC test case specified by the parameter from disk. If '-' is given instead of a file name, all test steps are cleared. Alternatively, several test steps may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '*', '?', and '[...]' and is evaluated just as in a UNIX command shell.

Clear-Observer

Parameters:

<Process type>

Defines all process instances of the given type to be usual SDL processes, not observer processes.

Clear-Parameter-Test-Values

Parameters:

```
( <Signal> | '-' ) ( <Parameter number> | '-' )
( <Value> | '-' )
```

The test value described by the value parameter is cleared for the signal parameter given as the parameter to the command. If the specified value parameter is '-', clears all test values for the signal parameter given as the parameter to the command. If '-' is given instead of the parameter number then the test values for all parameters of the signal are cleared. If '-' is given instead of the signal name then all test values for all parameters to all signals are cleared.

Regenerates the set of signals that can be sent from the environment during state space exploration.

Clear-Reports

Parameters:

```
(None)
```

Delete the current reports from the latest state space exploration.

Clear-Rule

Parameters:

```
(None)
```

The currently defined rule is deleted.

Clear-Signal-Definitions

Parameters:

```
<Signal> | '-'
```

Clears all currently defined test values for the signal given by the parameter. If '-' is given, the test values for all signals are cleared.

> **Note:**
>
> The signals cleared by this command may be regenerated if any of the commands for defining test values for sorts or parameters are used.

Clear-Test-Values

Parameters:

```
( <Sort> | '-' ) ( <Value> | '-' )
```

Clears all test values for the sort given as parameter. If the specified sort parameter is '-', all test values for all sorts will be cleared. If the value parameter is '-', all test values for the specified sort will be cleared.

Regenerates the set of signals that can be sent from the environment during state space exploration.

Command-Log-Off

Parameters:

```
(None)
```

The command log facility is turned off; see the command Command-Log-On for details.

Command-Log-On

Parameters:

```
<Optional file name>
```

The command enables logging of all the commands given in the monitor. The first time the command is entered a file name for the log file has to be given as parameter. After that any further Command-Log-On commands, without a file name, will append more information to the previous log file, while a Command-Log-On command with a file name will close the old log file and start using a new file with the specified name.

Initially the command log facility is turned off. It can be turned off explicitly by using the command Command-Log-Off.

The generated log file is directly possible to use as a file in the command Include-File. It will, however, contain exactly the commands given in the session, even those that were not executed due to command errors. The concluding Command-Log-Off command will also be part of the log file.

Continue-Until-Branch

Parameters:

```
<Optional node number>
```

First go to the system state node which is the child with the specified number to the current system state. Same as the Next command. Then go down as long as there is only one child, i.e. a branch is found.

Continue-Up-Until-Branch

Parameters:

(None)

Go up in the behavior tree as long as there is only one child, i.e. a branch is found.

Default-Options

Parameters:

(None)

Resets all options in the Validator to their default values and clears all reports. It also sets the current state to the current root. Compare with the command Reset.

Define-Autolink-Configuration

Parameters:

<Configuration>

Defines an Autolink configuration which might consist of:

1. Rules for the handling of constraints and signal parameter values (naming, replacement and parameterization).
2. Rules for the arrangement of test cases (test suite structure).
3. Rules for declaring ASP and PDU types.
4. A rule for stripping signal definitions from constraints.
5. Functions used by the rules 1 and 2 above.

If any rule or function is already defined, Autolink prompts the user to decide whether to add the new rules or to clear the existing Autolink configuration. If Define-Autolink-Configuration is part of a command file, the existing configuration is always replaced.

Rules beginning with TRANSLATE are evaluated when a constraint is created during test case generation. A translation rule specifies the name and the parameters of a constraint for a given SDL signal. In addition, it allows to replace signal parameter values by wildcards in a constraint

declaration table (TTCN matching mechanism). Translation rules can also be used to introduce test suite parameters and constants.

Test suite structure rules are evaluated when a test suite is saved. A test suite structure rule specifies a test group reference for test cases and/or test steps.

The two rules for the mapping of SDL sorts and signals onto ASP/PDU types are evaluated when writing the TTCN MP file. The same applies to the `STRIP-SIGNALS` rule that tells Autolink to create a constraint for a (single) parameter of an SDL signal instead for the signal itself.

An Autolink configuration has to obey the syntax rules described in "Autolink Configuration Syntax" on page 2314.

For a detailed description of the semantics, see "Syntax and Semantics of the Autolink Configuration" on page 1433 in chapter 36, *TTCN Test Suite Generation*.

Define-Autolink-Depth

Parameters:

```
<Depth>
```

Sets the maximum depth of the state space exploration for test case generation to the given value. The default value is 1000.

Define-Autolink-Generation-Mode

Parameters:

```
"Partial-Ordering" | "Total-Ordering"
```

Determines the way Autolink analyzes an MSC test case during test generation. If set to "Partial-Ordering", the standard semantics of MSC is used whereas "Total-Ordering" lets Autolink evaluate incoming and outgoing messages in an MSC straight from top to bottom. Typically, "Partial-Ordering" results in a behavior tree with several leaves whereas "Total-Ordering" produces only one sequence of events. The default setting is "Partial-Ordering". Note: The total ordering interpretation is not supported by the Generate-Test-Case command.

Define-Autolink-Hash-Table-Size

Parameters:

```
<Size>
```

Sets the size of the hash table used for test case generation to `<Size>` bytes. The default value is 1,000,000 bytes.

Define-Autolink-State-Space-Options

Parameters:

```
"On" | "Off"
```

Determines whether Autolink sets its own default state space options before generating test cases. If “On’, the state space options are automatically set when Generate-Test-Case is invoked and reset to the previous options when the command is completed. The default value is “On”.

Note that in general, the Validator’s default options are too strict to find all inconclusive events. Also be aware that for Autolink to work correctly, the priority of “Input from ENV” has to be higher than the priority of all other event classes, and the queues for all channels to the environment have to be activated.

Define-Bit-State-Depth

Parameters:

```
<Depth>
```

The parameter is the maximum depth of bit state exploration, i.e., the number of levels to be reached in the behavior tree. If this level is reached during the exploration, the current path is8 truncated and the exploration continues in another node of the behavior tree. The default value is 100.

Define-Bit-State-Hash-Table-Size

Parameters:

```
<Size in bytes>
```

Sets the size of the hash table used to represent the generated state space during bit state exploration. The default value is 1,000,000 bytes.

Define-Bit-State-Iteration-Step

Parameters:

```
<Step>
```

The Bit-State-Exploration algorithm includes a feature to automatically make a number of explorations with an increased depth for each exploration. The iteration continues until the search depth is greater than the maximum depth search as defined by Define-Bit-State-Depth command or one exploration terminates without any truncations.

This command activates the feature and defines how much the depth is increased for each iteration. If <Step> is set to 0 the iterative exploration

is deactivated, otherwise <Step> defines how much the maximum depth is increased for each exploration.

The default value is 0, i.e. the feature is not activated.

Define-Channel-Queue

Parameters:

```
<Channel name> ( "On" | "Off" )
```

Adds or removes a queue for the specified channel. If a queue is added for a channel, it implies that when a signal is sent that is transported on this channel, it will be put into the queue associated with the channel. By default no channels have queues.

Define-Concurrent-TTCN

Parameters:

```
"On" | "Off"
```

Defines whether TTCN test suites should be saved in concurrent TTCN format.

The default value is “off”.

> **Note:**
>
> Concurrent TTCN is not supported in combination with the “Local” or “Inline” TTCN test steps format. If an error message appears when trying to enable concurrent TTCN, first set the test steps format to “Global” with Define-TTCN-Test-Steps-Format.

Define-Condition-Check

Parameters:

```
"On" | "Off"
```

Defines whether conditions in an MSC should be used for synchronization purposes during MSC verification and test case generation. If condition check is turned on, then events below a global condition can only occur if all events on all instances above that condition have been processed; the condition becomes a synchronization point. If the condition is not global, then only the connected instances are synchronized. If condition check is turned off, then conditions in MSCs are ignored.

The default value is “off”.

Note:

If condition check is turned on, then the semantics of conditions is different from the semantics given in the ITU-T Recommendation Z.120!

Define-Constraint

Parameters:

```
<Constraint name> <Signal definition>
```

Defines a new constraint, which is automatically written to the test suite if the user enters the Save-Test-Suite command later.

Define-Exhaustive-Depth

Parameters:

```
<Depth>
```

The parameter defines the depth of the search when performing exhaustive exploration. The default value is 100.

Define-Global-Timer

Parameters:

```
"On" | "Off"
```

Defines whether a global timer should be added to TTCN test suites. If this option is enabled, the following things are done automatically:

- A declaration for a timer “T_Global” is generated with default duration “PIX_T_Global” and a unit “s” (seconds);
- A test suite parameter “PIX_T_Global” with type “INTEGER” is declared;
- A statement “START T_Global” is inserted at the beginning of each Test Case Dynamic Behaviour table; if concurrent TTCN output is enabled, a “START T_Global” is also inserted at the beginning of each top-level Test Step Dynamic Behaviour table for all parallel test components;
- A statement “CANCEL T_Global” is appended to every test sequence in each Test Case Dynamic Behaviour table; if concurrent TTCN output is enabled, a “CANCEL T_Global” is also added to the end of every test sequence in each top-level Test Step Dynamic Behaviour table of all parallel test components.

The default value is “on”.

Note:

It is recommended that the decision on using global timers is made before test case generation is started and that it is not changed later. Changing the option after the first test case has been generated may lead to unexpected results.

The name of the global timer cannot be changed. However, the default duration and unit may be defined with the Define-Timer-Declaration command. This should be done before the first test case is generated.

Define-Instance-Conversion

Parameters:

```
<From string> <To string>
```

Adds an entry into an instance conversion table. If a string contains spaces, it must be quoted.

When reading instances in an MSC diagram and the name textually match a <From string> in the conversion table, it will be replaced by the <To string>. This is useful if you want to verify some, but not all, instances in an MSC, in which case you can convert the unwanted instances to be considered as environment.

Note that this command will not change an already loaded MSC.

Define-Integer-Output-Mode

Parameters:

```
"dec" | "hex" | "oct"
```

Defines whether integer values are printed in decimal, hexadecimal or octal format. In hexadecimal format the output is preceded with “0x”, in octal format the output is preceded with ‘0’ (a zero).

On input: if the format is set to hexadecimal or octal, the string determines the base as follows: After an optional leading sign a leading zero indicates octal conversion, and a leading “0x” hexadecimal conversion. Otherwise, decimal conversion is used.

The default is “dec”, and no input conversion is performed.

Define-Max-Input-Port-Length

Parameters:

```
<Number>
```

The maximum length of the input port queues is defined. If this length is exceeded during state space exploration, a report is generated (see "Max Queue Length Exceeded" on page 2300). The default value is 3.

Define-Max-Instance

Parameters:

```
<Number>
```

Defines the maximum number of instances allowed for any particular process type. If this number is exceeded during state space exploration, a report is generated (see "Create Errors" on page 2299). The default value is 100.

Define-Max-Signal-Definitions

Parameters:

```
<No of signals>
```

This command defines the maximum no of signals that will be added to the list of signals from the environment for any particular signal. The default value is 10.

Define-Max-State-Size

Parameters:

```
<Size in bytes>
```

Sets the size of an internal array used to store each system state when computing the hash value for the system state. The default size is 100,000 bytes.

Define-Max-Test-Values

Parameters:

```
<No of text values>
```

This command defines the maximum no of test values that will be generated for a particular data type or signal parameter. The default value is 10.

Define-Max-Transition-Length

Parameters:

```
<Number>
```

The maximum number of SDL symbols allowed to be executed during the performance of a behavior tree transition is defined. If this number is exceeded during state space exploration, a report is generated (see "Transition Length Error" on page 2304). The default value is 1,000.

Define-MSC-Ignore-Parameters

Parameters:

```
"On" | "Off"
```

Defines whether the parameters of signals (and timers) will be ignored (i.e. set to default values) when reading MSC diagrams. The default is "Off".

Note that this command will not change an already loaded MSC.

Define-MSC-Search-Mode

Parameters:

```
"Verification" | "Violation"
```

Defines options for MSC verification depending on the purpose of the exploration. "Verification" mode should be used if the intention with the exploration is to verify all MSC alternatives. "Violation" mode should be used if the intention is to check if the SDL system can violate the MSC.

The difference between "Verification" and "Violation" mode is that in "Verification" mode the validator will:

- synchronize on all MSC starts and start of inline expressions, and
- perform a separate MSC action to enter an MSC / inline expression.

The synchronization in "Verification" mode implies that the validator will try to execute all events in one MSC before continuing with the events in a subsequent MSC. The special MSC action is executed when all events in one MSC has been executed and it is time to start with the subsequent MSC. Note that if there are more than one subsequent MSC (e.g. due to an HMSC alternative or an inline expression) each MSC action will select one of the MSCs. In the trace output (and in the Navigator) the MSC actions will be presented as "Enter MSC XXX" where XXX is the name of the MSC that the validator will try to execute next.

A simple way to get hands-on experience with the difference between "Verification" and "Violation" mode is to load an MSC that includes alternatives (either an HMSC or an MSC with inline expressions) using the Load-MSC command. Then set up the MSC search mode and use the Navigator to manually explore the state space.

Note that in "Verification" mode the **par** operator is not supported.

The default search mode is "Violation" mode.

Define-MSC-Test-Cases-Directory

Parameters:

```
<Directory name> | '-'
```

Defines the directory in which the MSC test cases are stored. If '-' is provided instead of a directory name, the test cases directory becomes undefined.

Define-MSC-Test-Steps-Directory

Parameters:

```
<Directory name> | '-'
```

Defines the directory in which the MSC test steps are stored. If '-' is provided instead of a directory name, the test steps directory becomes undefined.

Define-MSC-Trace-Action

Parameters:

```
"On" | "Off"
```

This command defines whether actions like tasks, decisions, etc. are shown in the MSC trace. The default value is "Off".

Define-MSC-Trace-Autopopup

Parameters:

```
"On" | "Off"
```

This command defines whether an MSC Editor automatically will pop up and show an MSC trace when going to a report. The default value is "On".

Define-MSC-Trace-State

Parameters:

```
"On" | "Off"
```

This command defines if the SDL process graph states are shown in the MSC trace. If "On", a condition symbol will be shown in the MSC trace each time a process performs a nextstate. The default value is "On".

Define-MSC-Trace-Channels

Parameters:

```
"On" | "Off"
```

Defines whether the env instance should be split into one instance for each channel connected to env in the MSC trace. The default is "Off".

Define-MSC-Verification-Algorithm

Parameters:

```
"TreeSearch" | "BitState"
```

Defines the search algorithm that is used for MSC verification. Two options are available, the tree search algorithm and the bit state algorithm. The default is "BitState".

Note that this command only defines what search algorithms that is used by the Verify-MSC command. It is possible to use other search algorithms, like Exhaustive-Exploration, by manually loading the MSC with the Load-MSC command and then starting the desired search algorithm.

Define-MSC-Verification-Depth

Parameters:

```
<Depth>
```

The parameter defines the depth of the search when performing MSC verification using the Verify-MSC command. The default value is 1,000.

Define-Observer

Parameters:

```
<Process type>
```

Defines all process instances of the given type to be observer processes.

Define-Parameter-Test-Value

Parameters:

```
<Signal> <Parameter number> <Value>
```

The parameter test value described by the command parameters is added to the current set of test values. The list of signals that can be sent from the environment is regenerated based on the new set of test values.

Define-Priorities

Parameters:

```
<Internal events> <Input from ENV> <Timeout events>
<Channel output> <Spontaneous transitions>
```

Defines the priorities for the different event classes. The priorities can be set individually to 1, 2, 3, 4 or 5. For more information about event classes and priorities, see "Event Priorities" on page 2397 in chapter 54, *Validating a System*.

The default priorities are:
- Internal events: 1
- Input from ENV: 2
- Timeout events: 2
- Channel outputs: 1
- Spontaneous transitions: 2

Note that the priorities of events processed in the validator also is affected by the command "Define-Symbol-Time" on page 2252.

Define-Random-Walk-Depth

Parameters:

```
<Depth>
```

The parameter defines the depth of the search when performing random walk exploration. The default value is 100.

Define-Random-Walk-Repetitions

Parameters:

```
<No of repetitions>
```

The parameter defines the number of times the search is performed from the start state when performing random walk exploration. The default is 100.

Define-Report-Abort

Parameters:

```
<Report type> | '-'
```

Defines that the state space exploration will be aborted whenever a report of the specified type is generated.

The available report types are listed with the command Show-Options. They are also described in "Rules Checked During Exploration" on page 2298. If the parameter is specified as '-', all report types will be defined in the same way.

Define-Report-Continue

Parameters:

```
<Report type> | '-'
```

Defines that a state space exploration will continue past a state where a report of the specified type is generated. With this definition, the exploration of the behavior tree is not affected by a report being generated.

The available report types are listed with the command Show-Options. They are also described in "Rules Checked During Exploration" on page 2298. If the parameter is specified as '-', all report types will be defined in the same way.

Define-Report-Log

Parameters:

```
<Report type> ( "Off" | "One" | "All" )
```

Defines how many reports of a specified type will be stored in the list of found reports when they are encountered during state space exploration. Default for all report types is "One".

If "Off" is specified for a report type, reports of this type will not be stored, which implies for example that they will not be listed by the List-Reports command. However, the reports will be generated and the appropriate action taken as specified by the commands Define-Report-Abort, Define-Report-Continue and Define-Report-Prune.

If "One" is specified only one occurrence of each reported situation is stored.

If "All" is specified then all occurrences of a reported situation that have different execution paths is stored in the list.

The available report types are listed with the command Show-Options. They are also described in "Rules Checked During Exploration" on page 2298. If the parameter is specified as '-', all report types will be defined in the same way.

Define-Report-Prune

Parameters:

```
<Report type> | '-'
```

Defines that a state space exploration will not continue past a state where a report of the specified type is generated. Thus, the part of the behavior tree beneath the state will not be explored. Instead, the exploration will continue in the siblings or parents of the state. This is also known as "pruning" the behavior tree at the state. This is the default behavior for all report types.

The available report types are listed with the command Show-Options. They are also described in "Rules Checked During Exploration" on page 2298. If the parameter is specified as '-', all report types will be defined in the same way.

Define-Report-Viewer-Autopopup

Parameters:

```
"On" | "Off"
```

This command defines whether the Report Viewer automatically will pop up when an automatic exploration is finished. The default is “On”.

Define-Root

Parameters:

```
"Original" | "Current"
```

Defines the root of the behavior tree to be either the current system state or the original start state of the SDL system.

> **Note:**
>
> When the root is redefined, all paths, e.g. MSC traces or report paths, will start in the new root, not in the original start state.

Define-Rule

Parameters:

```
<User-defined rule>
```

A new rule is defined that will be checked during state space exploration.

Define-Scheduling

Parameters:

```
"All" | "First"
```

Defines which process instances in the ready queue are allowed to execute at each state. The parameter defines the scheduling as follows:

- `All`
  All process instances in the ready queue are allowed to execute at each state.
- `First`
  Only the first process instance in the ready queue is allowed to execute at each state.

The default is “First”.

Define-Signal

Parameters:

```
<Signal> <Optional parameter values>
```

A signal that is to be sent to the SDL system from the environment is defined. The signal is defined by its name and optionally the values of its parameters. Multiple Define-Signal commands may be used to define the same signal, but with different values for the parameters.

Note:

The signals defined by this command will be destroyed if the signals are regenerated, i.e., if any of the commands defining test values for sorts or signal parameters are used.

Define-Spontaneous-Transition-Progress

Parameters:

```
"On" | "Off"
```

Defines whether a spontaneous transition (input none) is considered as progress when performing non-progress loop check. Default is that spontaneous transition is considered to be progress, i.e. “On”. See “Non Progress Loop Error” on page 2304.

Define-Symbol-Time

Parameters:

```
"Zero" | "Undefined"
```

The time it takes to execute one symbol, e.g. an input, task or decision, in an SDL process is defined either to be zero or undefined. If it is set to zero, it is assumed that all actions performed by process instances take are infinitely fast compared to the timer values that are used in the system. If the symbol time is set to undefined, no assumption is made about how long time it takes for processes to execute symbols. Consider for example a situation where a process sets a timer with a duration 5 and then executes something that may take a long time, e.g. a long loop, and the sets a timer with duration 1. If symbol time is set to zero then the second timer will always expire first. If symbol timer is set to undefined then both timers can potentially expire first.

Note that when symbol time is set to zero, no timer will expire if an internal action is possible, even if internal and timer events have the same priority as set by the command Define-Priorities.

The default value of the symbol time options is “Zero”.

Define-Test-Value

Parameters:

```
<Sort> <Value>
```

The test value described by the parameters is added to the current set of test values. The list of signals that can be sent from the environment is regenerated based on the new set of test values.

> **Note:**
>
> When regenerating the set of signals, all signals that have been manually defined using the Define-Signal command will be lost.

Define-Timer-Check-Level

Parameters:

```
<Number>
```

The action taken during MSC verification when checking timer statements (set, reset and timeout) is defined. The possible values for the specified number are:

- 0: No checking of timer events is performed.
- 1: If a timer event exists in the MSC a matching timer event must exist in the explored SDL path, but a timer event in the explored SDL path is accepted even if there is no corresponding MSC timer event.
- 2: All timer events in the MSC must match a corresponding timer event in the explored SDL path, and vice versa.

The default value is 1.

Define-Timer-Declaration

Parameters:

```
<Timer name> <Duration> <Unit>
```

Creates or updates a TTCN timer declaration. Autolink distinguishes between implicit timer declarations, which are created from test description MSCs during test case generation, and explicit timer declarations, which are created with this command. Valid explicit declarations always override implicit ones. Existing explicit declarations cannot be updated.

The duration may be an integer value or a valid test suite parameter identifier. If a test suite parameter is specified, a declaration for this pa-

rameter is generated as well. If no valid duration is specified, the duration field in the declaration remains empty.

The unit must be a valid TTCN unit: “ps”, “ns”, “us”, “ms”, “s” or “min”. No declaration is made if an invalid unit is specified.

Note:

It is recommended that all timers appearing in MSC test cases and test steps are declared explicitly prior to test case generation.

Define-Timer-Progress

Parameters:

```
"On" | "Off"
```

Defines if the expiration of a timer is considered as progress when performing non-progress loop check. Default is that timer expiration is considered to be progress, i.e. “On”. See “Non Progress Loop Error” on page 2304.

Define-Transition

Parameters:

```
"SDL" | "Symbol-Sequence"
```

Defines the semantics and length of the transitions in the behavior tree. The parameter defines the transitions as follows:

- `SDL`
  The behavior tree transitions correspond to complete SDL process graph transitions.
- `Symbol-Sequence`
  The behavior tree transitions correspond to the longest sequence of SDL symbols that can be executed without any interaction with other process instances.

The default is “SDL”.

Define-Tree-Search-Depth

Parameters:

```
<Depth>
```

Defines the maximum search depth for tree search.

The default value is 100.

Define-TTCN-Compatibility

Parameters:

"Standard" | "ITEX"

Determines whether Autolink generates the declarations part of a TTCN test suite. If set to “Standard”, Autolink creates declarations with ASN.1 data field names being constructed of the form “data type+parameter number” (e.g. “Integer2”). If set to “ITEX”, no declarations are created. In this case, ASN.1 data field names used in constraint definitions are represented by ‘#’ characters. This allows to merge the test suite with declarations generated by Link in the TTCN suite. The default value is “Standard”.

This command exists for backward compatibility reasons only.

Define-TTCN-Signal-Mapping

Parameters:

"ASP" | "PDU"

Determines whether SDL signals are mapped onto ASN.1 ASP constraints or ASN.1 PDU constraints. By default, signals are mapped onto ASPs. The setting of Define-TTCN-Signal-Mapping can be overruled for single signals by the use of the ’ASP-Types’ and ’PDU-Types’ commands in the Autolink configuration language (cf. Define-Autolink-Configuration).

Define-TTCN-Test-Steps-Format

Parameters:

"Global" | "Local" | "Inline"

Defines how test steps are written in a TTCN test suite. “Global” is the default value.

If set to “Global”, each test step is saved in a separate behaviour description table. If a test step is used in more than one test case, then only one behaviour description table is generated.

If set to “Local”, each test step is saved as a local tree in the behaviour table of the corresponding test case. If a test step is used several times in a single test case, then only one behaviour description is generated.

If set to “Inline”, all test step trees are directly inserted into the test case descriptions.

Note that a test step which is used in several places may lead to trees with different inconclusive events or different verdicts. In this case, the test steps are stored in separate tables with unique names.

> **Note:**
>
> Test step formats “Local” and “Inline” are not supported in combination with concurrent TTCN. If an error message appears when trying to change the test steps format, first disable concurrent TTCN with Define-Concurrent-TTCN.

Define-Variable-Mode

Parameters:

```
<Process or Process Type> [ <Variable name> |
“Parent” | “Offspring” | “Sender” ]
[ “Compare” | “Skip” ]
```

This command defines how a specific variable is treated when comparing two system states during state space exploration. If the value for a variable is “Compare”, this variable will be taken into account when comparing two system states. If the value is “Skip”, this variable will not be taken into account, i.e. if the only difference between two system states is that values of variables in “Skip” mode differs, then the system states will be considered equal.

The purpose of the “Skip” mode for variables is to optimize the state space search. There are two different situations where this command can be used:

- All variables that is known to be constant during an exploration can be declared “Skip”.
- All variables that will not have an effect on the dynamic behavior of the system, i.e. that will not affect the path through a decision or the expression in an “output to”, can be declared “Skip”.

The benefit with constant variables in “Skip” mode is that the Validator will ignore these variables when computing hash values. This can for large data structures like arrays mean that the performance of the validator can be considerably improved.

Detailed-Exa-Var

Parameters:

```
(None)
```

When printing structs containing components with default value, these values are explicitly printed after this command is given.

Down

Parameters:

```
<Number of levels>
```

Go down the specified number of levels in the behavior tree, each time selecting the child of the current system state that is part of the current path. If the parameter is too large, Down will stop at the end of the current path.

Evaluate-Rule

Parameters:

```
(None)
```

The currently defined rule is evaluated with respect to the current system state. The command prints whether or not the rule is satisfied.

Examine-Channel-Signal

Parameters:

```
<Channel name> <Entry number>
```

The parameters of the signal instance at the position equal to the entry number in the queue of the specified channel are printed. The entry number is the number associated with the signal instance when the command List-Channel-Queue is used.

Examine-PId

Parameters:

```
(None)
```

Information about the process instance given by the current scope is printed (see the Set-Scope command for an explanation of scope). This information contains the current values of Parent, Offspring, Sender and a list of all currently active procedure calls made by the process instance. The list starts with the latest procedure call and ends with the process instance itself.

If the process instance contains service(s), information about all services is also printed.

Examine-Signal-Instance

Parameters:

```
<Entry number>
```

The parameters of the signal instance at the specified position in the input port of the process instance given by the current scope are printed (see the Set-Scope command for an explanation of scope). The entry number is the number associated with the signal instance when the command List-Input-Port is used.

Examine-Timer-Instance

Parameters:

```
<Entry number>
```

The parameters of the specified timer instance are printed. The entry number is the number associated with the timer when the List-Timer command is used.

Examine-Variable

Parameters:

```
[ '(' <PId value> ')' ]
<Optional variable name>
<Optional component selection>
```

The value of the specified variable or formal parameter in the current scope is printed (see the Set-Scope command for an explanation of scope). Variable names may be abbreviated. If no variable name is given, all variable and formal parameter values of the process instance given by the current scope are printed. Sender, Offspring, and Parent may also be examined in this way. Their names, however, may not be abbreviated and they are not included in the list of all variables.

> **Note:**
>
> If a variable is exported, both its current value and its exported value are printed.

It is possible to examine only a component of a struct, string or array variable, by appending the struct component name or a valid array index value as an additional parameter. The component selection can handle structs and arrays within structs and arrays to any depth by giving a list component selection parameters. SDL syntax with '!' and "( )" as well as just spaces, can be used to separate the names and the index values.

It is also possible to print a range of an array by giving “FromIndex : ToIndex” after an array name. Note that the space before the ‘:’ is required if FromIndex is a name (enumeration literal), and that no further component selection is possible after a range specification.

To see the possible components that are available in the variable, the variable name must be appended by a space and a ‘?’ on input. A list of components or a type name is then given, after which the input can be continued. After a component name, it is possible to append a ‘?’ again to list possible sub components.

To print the value of the data referenced by a Ref pointer it is possible to use the SDL syntax, i.e. the “*>” notation. If, for example, `Iref` is a Ref(Integer) variable, then `Iref*>` is the integer referenced by this pointer. If `Sref` is a Ref of a struct, then `Sref*> ! Comp1` is the Comp1 component in the referenced struct. The sequence `*> !` can in the monitor be replaced by `->` (as for example in C).

If a PId is given within parenthesis, the scope is temporarily changed to this process instance instead.

Exhaustive-Exploration

Parameters:

`(None)`

Starts an automatic state space exploration from the current system state, where the entire generated state space is stored in primary memory. This is only recommended for SDL systems with small state spaces.

The exploration will continue until either the complete state space to the defined depth is explored, `<Return>` is pressed from the command prompt, or the *Break* button is pressed in the graphical user interface. The system is then returned to its initial system state. The maximum depth of the exploration can be set with the command Define-Exhaustive-Depth.

If an exhaustive exploration already has been started, but has been stopped, this command asks if the exploration should continue from where it was stopped, or restart from the beginning again.

A status message is printed every 50,000 states that are generated. When the exploration is finished or stopped, the same information as for a bit state exploration is printed.

Exit

Parameters:

(None)

The executing validator is terminated. If the command is abbreviated, the monitor asks for confirmation. If any of the Validator options have been changed, the monitor will ask if the changed options should be saved. If so, the changed options are saved in a file `.valinit` **(on UNIX)**, or `valinit.com` **(in Windows)**, in the directory from where the SDL suite was started. This file is automatically loaded the next time a Validator is started from the same directory, thus restoring the previously saved options.

This is the same command as Quit.

Extract-Signal-Definitions-From-MSC

Parameters:

<Directory name> (<File name> | '-')

Extracts all signals from an MSC which are to be sent from the environment into the SDL system and adds them to the list of active signals. This command is only applicable to basic MSCs in textual format, i.e. to those MSCs stored with file extension `.mpr`. If '-' is given instead of a file name, signal definitions from all MSCs in the specified directory are extracted. Alternatively, several MSCs may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '`*`', '`?`', and '`[...]`' and is evaluated just as in a UNIX command shell.

Generate-Test-Case

Parameters:

<File name> | '-'

Starts the state space search algorithm which constructs an internal data structure for the MSC test case specified by the parameter. Additionally, constraints for all signals occurring in the test case are generated. At the end all constraints with identical signal definitions are merged.

If the command parameter is specified as '`-`', test cases are generated for all MSC test cases in the current test cases directory. Alternatively, several MSC test cases may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '`*`', '`?`', and '`[...]`' and is evaluated just as in a UNIX command shell.

If the previous generation of a test case has been interrupted, Autolink prompts the user to decide whether to continue the interrupted generation or to abort it. No file name is needed if the test case generation is to be continued.

Note that this command requires a TTCN Link license in addition to the Validator license.

Goto-Path

Parameters:

```
<Path>
```

Go to the system state specified by the path. For details about paths, see the command Print-Path.

Goto-Report

Parameters:

```
<Report number>
```

Go to the state in the behavior tree where the report with the corresponding number has been found. The last behavior tree transition that was executed before the reported situation is printed, with the same information as for a full trace during simulation. The report number is the number associated with the report when the command List-Reports is used. If only one report exists, the report number is optional.

Help

Parameters:

```
<Optional command name>
```

Issuing the Help command without a parameter will print all the available commands. If a command name is given as parameter, this command will be explained.

Include-File

Parameters:

```
<File name>
```

This command provides the possibility to execute a sequence of monitor commands stored in a text file. The Include-File facility can be useful for including, for example, an initialization sequence or a complete test case. It is allowed to use Include-File in an included sequence of commands; up to five nested levels of include files can be handled.

List-Channel-Queue

Parameters:

```
<Channel name>
```

A list of all signal instances in the specified channel queue is printed. For each signal instance an entry number, the signal type, and the sending process instance is given. The entry number can be used in the command Examine-Channel-Signal.

List-Constraints

Parameters:

```
(None)
```

Lists all currently defined constraints.

List-Generated-Test-Cases

Parameters:

```
(None)
```

Lists all test cases which have been generated either by Generate-Test-Case or Translate-MSC-Into-Test-Case, or which have been loaded with the Load-Generated-Test-Cases command.

List-Input-Port

Parameters:

```
[ '(' <PId value> ')' ]
```

A list of all signal instances in the input port of the process instance given by the current scope is printed (see the Set-Scope command for an explanation of scope). For each signal instance an entry number, the signal type, and the sending process instance is given. A '*' before the entry number indicates that the corresponding signal instance is the signal instance that will be consumed in the next transition performed by the process instance. The entry number can be used in the command Examine-Signal-Instance.

If a PId is given within parenthesis information about this process instance is printed instead.

List-Instance-Conversion

Parameters:

```
(None)
```

Lists all defined instance conversions. See also the command Define-Instance-Conversion.

List-MSC-Test-Cases-And-Test-Steps

Parameters:

```
(None)
```

Lists all MSC test cases and test steps currently stored in the test cases and test steps directories.

List-Next

Parameters:

```
(None)
```

A list of the possible behavior tree transitions that can follow from the current system state is printed.

> **Note:**
>
> The number of possible transitions depends on the selected state space options.

List-Observers

Parameters:

```
(None)
```

List all process types which are defined to be observer processes.

List-Parameter-Test-Values

Parameters:

```
(None)
```

Lists all currently defined test values for signal parameters.

List-Process

Parameters:

```
<Optional process name>
```

A list of all process instances with the specified process name is printed. If no process name is specified all process instances in the system are listed. The list will contain the same details as described for the List-Ready-Queue command.

If the process contains services, information about the currently active service is also printed. To get information about all services use the Examine-PId command.

List-Ready-Queue

Parameters:

(None)

A list of process instances in the ready queue is printed. For more information, see the Simulator command "List-Ready-Queue" on page 2089 in chapter 50, *The SDL Simulator*.

List-Reports

Parameters:

(None)

All situations that have been reported during state space exploration are printed. For each report, the error or warning message describing the situation is printed, together with the depth in the behavior tree where it first occurred. Only one occurrence of each reported situation is printed; the one with the shortest path from the root of the behavior tree. The report numbers printed can be used in the command Goto-Report.

List-Signal-Definitions

Parameters:

(None)

A list of all currently defined signals is printed.

List-Test-Values

Parameters:

(None)

Lists all test values that currently are defined.

List-Timer

Parameters:

(None)

A list of all currently active timers is printed. For each timer, its corresponding process instance and associated time is given. An entry number will also be part of the list, which can be used in the command Examine-Timer-Instance.

Load-Constraints

Parameters:

<File name>

Loads all constraints stored in the file specified by the parameter. The suggested file extension is .con.

Load-Generated-Test-Cases

Parameters:

```
<File name>
```

Loads all generated test cases stored in the file specified by the parameter. The suggested file extension is `.gen`.

Load-MSC

Parameters:

```
<MSC file name>
```

The specified Message Sequence Chart is loaded into the Validator. Loading an MSC always resets the state space and sets the current state to the root of the behavior tree.

Load-Signal-Definitions

Parameters:

```
<File name>
```

A command file with Define-Signal commands is loaded and the signals are defined. This command exists for backward compatibility reasons only.

Log-Off

Parameters:

```
(None)
```

The command Log-Off turns off the interaction log facility, which is described in the command Log-On.

Log-On

Parameters:

```
<Optional file name>
```

The command Log-On takes an optional file name as a parameter and enables logging of all the interaction between the Validator and the user that is visible on the screen. The first time the command is entered, a file name for the log file has to be given as parameter. After that any further Log-On commands, without a file name, will append more information to the previous log file, while a Log-On with a file name will close the old log file and start using a new file with the specified file name.

Initially the interaction log facility is turned off. It can be turned off explicitly by using the command Log-Off.

Merge-Constraints

Parameters:

```
<First name> <Second name>
```

Merges the two constraints specified by the parameters. The resulting constraint has at least the formal parameters of the original constraints. If additional constraint parameters are necessary (e.g. when merging two constraints with signals S(1,1) and S(1,2)), Autolink prompts the user to enter a name for each new formal parameter. Merge-Constraints can only be applied to constraints with the same signal. The new constraint gets the name of the second constraint.

Merge-Report-File

Parameters:

```
<File name>
```

An existing report file is opened and the reports in it are added to the current reports.

MSC-Log-File

Parameters:

```
<File name>
```

A log file is produced containing information about the Message Sequence Chart for the trace from the root of the behavior tree to the current system state. This file can be opened in an MSC Editor; we recommend using a filename with the suffix `.mpr`, since that suffix is used as default.

MSC-Trace

Parameters:

```
(None)
```

An MSC Editor is opened, showing the Message Sequence Chart for the current execution path. The current state is shown in the MSC by the selection of some of the symbols. Whenever the current state and/or current path changes in the Validator, the MSC will be updated.

If the MSC trace is already on, this command will turn it off.

New-Report-File

Parameters:

```
<File name>
```

A new report file for report storage is created. The current reports are deleted.

Next

Parameters:

```
<Transition number>
```

Go to a system state in the next level of the behavior tree, i.e. a child to the current system state. The parameter is the transition number given by the List-Next command.

Open-Report-File

Parameters:

```
<File name>
```

An existing report file is opened and the reports in it are loaded. The current reports are deleted.

Parameterize-Constraint

Parameters:

```
<Constraint name> <Replacement definition>
```

Replaces concrete values in a constraint with formal parameters. The replacement is defined by a sequence of pairs of a signal parameter number followed by the corresponding formal parameter name. The definition must be terminated with a 0.

Print-Autolink-Configuration

Parameters:

```
(None)
```

Displays the current Autolink configuration on the screen.

Print-Evaluated-Rule

Parameters:

```
(None)
```

The currently defined rule is printed with the values obtained from the last evaluation of the rule. The printed information is in the form of a so-called parse tree, and may require some knowledge of such structures to be interpreted correctly.

Print-File

Parameters:

```
<File name>
```

The content of the named text file is displayed on the screen.

Print-Generated-Test-Case

Parameters:

```
<Test case name> | '-'
```

Displays the internal representation of the generated test case specified by the parameter on the screen. If '-' is given instead of a test case name, all generated test cases are shown. Alternatively, several test cases may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '*', '?', and '[...]' and is evaluated just as in a UNIX command shell.

Print-MSC

Parameters:

```
(None)
```

A textual version of the currently loaded MSC is printed. The format is similar to the textual MSC format defined in ITU recommendation Z.120.

Print-Path

Parameters:

```
(None)
```

The path to the current system state is printed. A path is a sequence of integer numbers, terminated with a 0, describing how to get to the current system state. The first number indicates what transition to select from the root, the second number what transition to choose from this state, etc. To go to the state specified by a path, use the command Goto-Path.

Print-Report-File-Name

Parameters:

```
(None)
```

The name of the current report file is printed.

Print-Rule

Parameters:

```
(None)
```

The currently defined rule is printed.

Print-Trace

Parameters:

`<Number of levels>`

The textual trace leading to the current system state is printed. The same information as for a full trace during simulation is printed for each behavior tree transition. The parameter determines how many levels up the trace should start. For example, Print-Trace 10 will print the ten last transitions.

Quit

Parameters:

`(None)`

The executing validator is terminated. If the command is abbreviated, the monitor asks for confirmation. If any of the Validator options have been changed, the monitor will ask if the changed options should be saved. If so, the changed options are saved in the file `.valinit` **(on UNIX)**, or `valinit.com` **(in Windows)**, in the directory from where the SDL suite was started. This file is automatically loaded the next time a Validator is started from the same directory, thus restoring the previously saved options.

This is the same command as Exit.

Random-Down

Parameters:

`<Number of levels>`

Go down the specified number of levels in the behavior tree, each time selecting a random child of the current system state.

Random-Walk

Parameters:

`(None)`

This command will perform an automatic exploration of the state space from the current system state. Random walk is based on the idea that if more than one transition is possible in a particular system state, one of them will be chosen at random. When the exploration is started, the following information is printed:

- Search depth: The maximum depth to walk down. This can be set with the command Define-Random-Walk-Depth.

- Repetitions: The number of random walks to perform from the start state. This can be set with the command Define-Random-Walk-Repetitions.

The exploration will continue until either it is finished, `<Return>` is pressed from the command prompt, or the *Break* button is pressed in the graphical user interface. The system is then returned to the state it was in before the exploration was started.

When the exploration is finished or stopped, the Report Viewer is by default opened (this can be changed with the command Define-Report-Viewer-Autopopup). A few statistics are also printed; see the command Bit-State-Exploration for an explanation of these.

Rename-Constraint

Parameters:

`<Old name> <New name>`

Renames a constraint. If a constraint with the same "new" name but a different signal definition already exists, the user is prompted to choose between one of the following operations (which can be abbreviated):

- `Rename`: A different name can be chosen for the constraint.
- `Keep_old`: The definition of the renamed constraint is cleared. Any references from previously generated test cases to the renamed constraint are redirected to the old constraint.
- `Overwrite_old`: The definition of the old constraint is cleared. Any references from previously generated test cases to the old constraint are redirected to the renamed constraint.

Restriction: If one of the constraints contains formal parameters, then a different name must always be chosen for the renamed constraint. Rename-Constraint in combination with its options `Keep_old` and `Overwrite_old` allows to merge two constraints, e.g. if some signal parameters are irrelevant (see also the command Merge-Constraints and consider the use of `CONSTRAINT MATCH` ... statements in an Autolink configuration).

Reset

Parameters:

`(None)`

Resets the state of the Validator to its initial state. This command resets all options and test values in the Validator to their initial values and clears all reports, MSCs, and user-defined rules. It also sets the current state and the current root to the original start system state.

This command reads the `.valinit` **(on UNIX)**, or `valinit.com` **(in Windows)**, file (see the command Exit). It is equivalent to closing the Validator and starting it again. Compare with the command Default-Options.

Save-As-Report-File

Parameters:

`<File name>`

The current reports are saved in a new file. The name of the current report file is set to the new file.

Save-Autolink-Configuration

Parameters:

`<File name>`

Saves the current Autolink configuration in a command file. The command name `Define-Autolink-Configuration` and the terminating `End` are written at the beginning and the end of the file in order to allow to reload the configuration with Include-File. The suggested file extension is `.com`.

Save-Constraint

Parameters:

`( <Constraint name> | '-' ) <File name>`

Saves the constraint in a file identified by `<File name>`. If '-' is given instead of a constraint name, all currently defined constraints are saved. The suggested file extension is `.con`.

Save-Coverage-Table

Parameters:

`<File name>`

Test coverage information is saved in the specified file. The test coverage table consists of two parts, a Profiling Information section, and a Coverage Table Details section. This is the same type of file as generated by the Simulator; for more detailed information about the file, see

"Print-Coverage-Table" on page 2095 in chapter 50, *The SDL Simulator*.

Save-Error-Reports-As-MSCs

Parameters:

```
(None)
```

Saves all reports (except those indicating an MSC verification) as MSCs in the target directory. The reports are stored as complete MSCs with file extension `.mpr`.

For each MSC a generic name consisting of the name of the currently loaded MSC (or "ErrorReport", if none is loaded), a four letter abbreviation of the error type and a sequence number is generated.

Save-Error-Reports-As-MSCs helps to find out what went wrong when running a batch job with a large number of MSC verifications.

Note: This command does not generate MSCs for MSC violation reports if the current MSC could be verified.

Save-Generated-Test-Case

Parameters:

```
( <Test case name> | '-' ) <File name>
```

Saves the generated test case in a file. If '-' is given instead of a test case name, all generated test cases which are currently stored in memory are saved. Alternatively, several test cases may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '`*`', '`?`', and '`[...]`' and is evaluated just as in a UNIX command shell.The suggested file extension is `.gen`.

This command allows to compute test cases in parallel on different machines and to merge them afterwards by using Load-Generated-Test-Cases.

Save-MSC-Test-Case

Parameters:

```
( <Test case name> | '*' )
```

Saves a test case as a system level Message Sequence Chart in the test cases directory (with file extension `.mpr`). The MSC contains a separate instance axis for each channel to the environment and another axis for the system. It comprises all signals sent to and from the environment for the trace from the root of the behavior tree to the current system

state. If '*' is given instead of a test case name, a generic test case name is generated.

Save-MSC-Test-Step

Parameters:

```
( <Test step name> | '*' )
```

Saves a test step as a system level Message Sequence Chart in the test steps directory (with file extension .mpr). The MSC contains a separate instance axis for each channel to the environment and another axis for the system. It comprises all signals sent to and from the environment for the trace from the root of the behavior tree to the current system state. If '*' is given instead of a test step name, a generic test step name is generated.

Save-Options

Parameters:

```
<File name>
```

Creates a Validator command file with the name given as parameter. The file contains commands defining the options of the Validator. If this file is loaded (using the command Include-File) the options will be restored to their saved values.

Save-Reports-as-MSC-Test-Cases

Parameters:

```
( <Report type> | '-' ) <Name prefix>
```

Saves all reports of a specified type as MSC test cases in the test cases directory (in system level MSC format; with file extension .mpr). The MSCs contain a separate instance axis for each channel to the environment and another axis for the system. For each MSC test case, a generic name is generated consisting of the name prefix, a four letter abbreviation of the report type and a five digit number. If '-' is provided instead of a report type, all current reports are saved.

Save-State-Space

Parameters:

```
<File name>
```

A Labelled Transition System (LTS) representing the generated state space is saved to a file. For a description of the file syntax, see section "State Space Files" on page 2316.

Note:

It is necessary to have executed an Exhaustive-Exploration command before the state space can be saved on a file.

Save-Test-Suite

Parameters:

```
<Test suite name> <File name>
```

Saves the generated test cases and all constraints in a test suite file in TTCN MP format. In addition, it generates the declarations part of the TTCN test suite. If the generation of a test case has been interrupted, Autolink prompts the user to decide whether to cancel the command or to abort the test case generation. The suggested file extension for the second parameter is `.mp`.

Note that this command requires a TTCN Link license in addition to the Validator license.

Save-Test-Values

Parameters:

```
<File name>
```

A command file is generated containing Validator commands that, if loaded with the Include-File command, will recreate the current test value definitions.

Scope

Parameters:

```
(None)
```

This command prints the current scope. See the command Set-Scope for a description of scope.

Scope-Down

Parameters:

```
<Optional service name>
```

Moves the scope one step down in the procedure call stack. If the current scope is a process containing services, one of the services should be specified. See also the commands Stack, Set-Scope and Scope-Up.

Scope-Up

Parameters:

```
(None)
```

Moves the scope one step up in the procedure call stack. Up from a service leads to the process containing the service. See also the commands Set-Scope, Stack and Scope-Down.

SDL-Trace

Parameters:

(None)

This command opens an SDL Editor that will show the transition leading to the current system state. Whenever the current system state changes in the Validator, the SDL Editor will be updated.

If the SDL trace already is on, the command will turn it off.

Set-Application-All

Parameters:

(None)

The state space options of the Validator are set to perform state space exploration according to the semantics of an application generated by the SDL Code Generator. No assumptions are made about the performance of the SDL system compared to timeout values or the performance of the environment.

This command sets the exploration mode factors by executing the following commands:

```
Define-Transition SDL
Define-Scheduling First
Define-Priorities 1 1 1 1 1
```

Set-Application-Internal

Parameters:

(None)

The state space options of the Validator are set to perform state space exploration according to the semantics of an application generated by the SDL Code Generator. The assumption is made that the time it takes for the SDL system to perform internal actions is very small compared to timeout values and the response time of the environment.

This command sets the exploration mode factors by executing the following commands:

```
Define-Transition SDL
Define-Scheduling First
Define-Priorities 1 2 2 1 2
```

Set-Scope

Parameters:

```
<PId value> <Optional service name>
```

This command sets the scope to the specified process, at the bottom procedure call. If the process contains services, one of the services can be given as parameter to the command. A scope is a reference to a process instance, a reference to a service instance if the process contains services, and possibly a reference to a procedure instance called from this process/service. The scope is used for a number of other commands for examining the local properties of a process instance. The scope is automatically set to the process that executed in the transition leading to the current system state.

See also the commands Scope, Stack, Scope-Down and Scope-Up.

Set-Specification-All

Parameters:

```
(None)
```

The state space options of the Validator are set to perform state space exploration according to the ITU semantics for SDL. No assumptions are made about the performance of the SDL system compared to timeout values, or the performance of the environment.

This command sets the exploration mode factors by executing the following commands:

```
Define-Transition Symbol-Sequence
Define-Scheduling All
Define-Priorities 1 1 1 1 1
```

Set-Specification-Internal

Parameters:

```
(None)
```

The state space options of the Validator are set to perform state space exploration according to the ITU semantics for SDL. The assumption is made that the time it takes for the SDL system to perform internal actions is very small compared to timeout values and the response time of the environment.

This command sets the exploration mode factors by executing the following commands:

```
Define-Transition Symbol-Sequence
Define-Scheduling All
```

```
Define-Priorities 1 2 2 1 2
```

Show-Coverage-Viewer

Parameters:

(None)

This command starts the Coverage Viewer. The current test coverage is automatically loaded and a symbol coverage tree is presented. See chapter 48, *The SDL Coverage Viewer* for further description of the Coverage Viewer.

Show-Mode

Parameters:

(None)

This command displays a summary of the current execution mode and some other information about the current state of the SDL Validator.

Show-Navigator

Parameters:

(None)

This command starts the Navigator tool to simplify interactive navigation in the behavior tree of the SDL system. See "The Navigator Tool" on page 2292 for more details.

Show-Options

Parameters:

(None)

The values of all options defined for the Validator are printed, including the report action defined for each report type.

Show-Report-Viewer

Parameters:

(None)

This command starts the Report Viewer that presents a hierarchical view of all reports generated during state space explorations. See "The Report Viewer" on page 2295 for more details.

Show-Versions

Parameters:

(None)

The versions of the SDL to C Compiler and the runtime kernel that generated the currently executing program are presented.

Signal-Disable

Parameters:

```
<Signal> | '-'
```

Disables all test values defined for the given signal. '-' means *all signals*. Use Signal-Enable to start using them again. See also Channel-Disable. Test values for a signal are only used if both the signal and the channel that transports the signal are enabled. By default, all signals and channels are enabled.

Signal-Enable

Parameters:

```
<Signal> | '-'
```

Enables all test values defined for the given signal. '-' means *all signals*. See also Channel-Enable. Test values for a signal are only used if both the signal and the channel that transports the signal are enabled. By default, all signals and channels are enabled.

Signal-Reset

Parameters:

```
<Signal> | '-'
```

Removes all existing test values for the given signal, and defines a default set of test values instead, using the current test value settings for data types and signal parameters. '-' means *all signals*.

Stack

Parameters:

```
(None)
```

The procedure call stack for the PId/service defined by the scope is printed. For each entry in the stack, the type of instance (procedure/process/service), the instance name and the current state is printed. See also the commands Set-Scope, Scope-Down and Scope-Up.

Top

Parameters:

```
(None)
```

Go up in the behavior tree to the start of the current path (the root system state).

Translate-MSC-Into-Test-Case

Parameters:

```
( <File name> | '-' )
```

Translates an MSC into an internal test case data structure without performing a state space search. Additionally, constraints for all signals occurring in the test case are generated. At the end, all constraints with identical signal definitions are merged.

If the command parameter is specified as '-', all MSCs in the current test cases directory are transformed into test cases. Alternatively, several MSC test cases may be selected by a pattern enclosed in apostrophes. Such a pattern may include the special characters '*', '?', and '[...]' and is evaluated just as in a UNIX command shell.

Translate-MSC-Into-Test-Case does not require a fully specified SDL system. However, the signals used in the MSC must be defined in the SDL specification. Additionally, the channels between the SDL system and its environment have to be specified in order to find out which MSC instances relate to PCOs and which relate to the SDL system. Therefore, a direct translation cannot be applied to MSCs which consist of only a single instance axis for the environment.

Since the translation is done statically, it is not checked whether the MSC can in fact be verified. In addition, no inconclusive events are computed, unless they are stated explicitly by the use of an MSC exception expression.

Note that this command requires a TTCN Link license in addition to the Validator license.

Tree-Search

Parameters:

```
(None)
```

Performs a tree search of the state space from the current system state. A tree search is an exploration where all possible combinations of actions are executed. The tree that is explored is exactly the same tree that can manually be inspected using the Navigator feature (or manual exploration using the Next / List-Next commands).

The depth of the tree search is bounded and is defined by the Define-Tree-Search-Depth command. The default depth is 100.

The command can be aborted by the user by pressing the *Break* button in the Validator UI or any key if running the validator from the command prompt.

Tree-Walk

Parameters:

```
<Timeout> <Coverage>
```

Performs an automatic exploration of the state space starting from the current system state. In contrast to Random-Walk, Tree-Walk is based on a deterministic algorithm that performs a sequence of tree searches with increasing depth starting at various states in the reachability graph. Tree Walk combines the advantages of both the depth-first and breadth-first search strategy - it is able to visit states located deep in the reachability graph and to find a short path to a particular state at the same time. Tree Walk is guided by a symbol coverage heuristic. Therefore it is particularly suitable for automatic test case generation.

Computation stops if either time exceeds <Timeout> (specified in minutes) or the targeted coverage (specified in percent) is reached. Alternatively, the exploration can be stopped at any time by pressing *Return* at the command prompt or by pressing the *Break* button if the graphical user interface is used.

Tree Walk creates a number of “TreeWalk” reports. These reports can be used for the generation of MSC test cases by applying command Save-Reports-as-MSC-Test-Cases.

Up

Parameters:

```
<Number of levels>
```

Go up the specified number of levels in the behavior tree. Up 1 goes to the parent state of the current system state. Up will stop at the root of the behavior tree (the start state) if the parameter is too large.

Verify-MSC

Parameters:

```
<MSC file>
```

This command will perform an automatic exploration from the current system state to search for all execution traces that are consistent with the MSC that is given as a parameter. The search algorithm is a variant of the bit state algorithm that is adapted to suite MSC verification. Before

the exploration is started, the root of the behavior tree is set to the current system state.

The file can either be an basic MSC file (`.msc`), a textual MSC file (`.mpr`) or a high-level MSC file (`.mrm`).

When the exploration is finished, the same information as for a bit state exploration is printed. In addition, a message is printed stating whether or not the MSC was verified, i.e. if an execution path was found that satisfied the MSC, and reports for MSC verification and MSC violation are generated.

If the MSC is a high-level MSC or it contains MSC reference expressions more that one MSC verification report can be generated. There will be one report for each sequence of 'leaf' MSCs that can be verified. For more information on criteria for MSC verification, see "MSC Verification Errors" on page 2305.

If the MSC was verified and exactly one MSC verification report was generated, the system state of the Validator will be set up to the last system state in the path that satisfied the MSC. Otherwise, the system is returned to its initial state.

If the MSC contains MSC references the Validator needs to match the logical MSC names in the MSC references with MSC file names. This is done according to the following algorithm:

1. If the Autolink test step directory is set, then the MSC file is assumed to be located in this directory and have the same name as the MSC with the suffix `.msc`, `.mpr` or `.mrm`.
2. The Validator then checks if there is an MSC with the correct name in the same module in the Organizer as the parent MSC. If this is the case this MSC file is used.
3. If none of above applies then the Validator checks if there is an MSC file in the same directory as the parent MSC with the correct name and with the suffix `.msc`, `.mpr` or `.mrm`. If this is the case this MSC file is used.

Graphical User Interface

This section describes the appearance and functionality of the graphical user interface to the validator monitor (ValUI). Some user interface descriptions general to all tools in Telelogic Tau can be found in chapter

1, *User Interface and Basic Operations*. These general descriptions are not repeated in this chapter.

Note:

The ValUI looks and behaves very much like the SimUI, the graphical user interface to the simulator monitor. The differences are the set of available monitor commands and button modules, and a few additional menus and sub windows.

For an explanation of the Validator main window, depicted in Figure 474, see "Graphical User Interface" on page 2130 in chapter 50, *The SDL Simulator*.

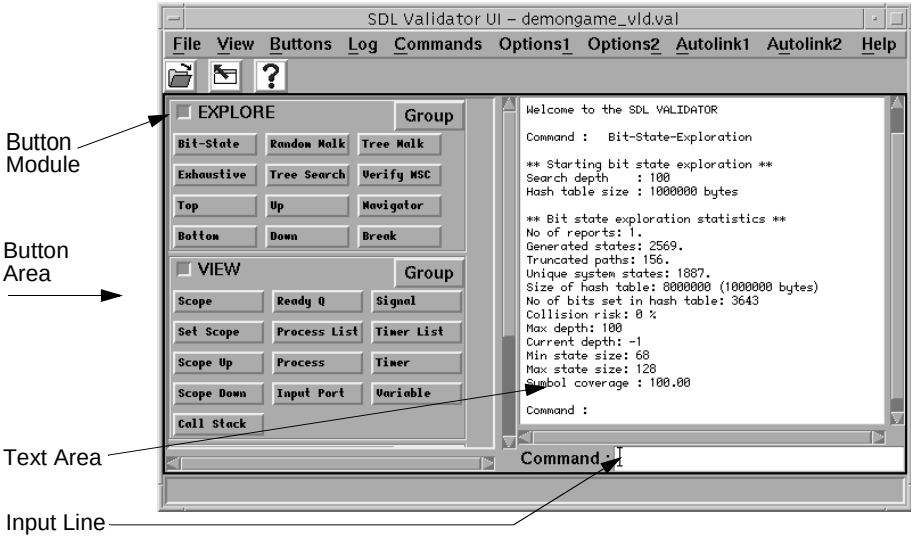

Figure 474: The main window

Starting the ValUI

A new ValUI is started by selecting *SDL > Validator UI* from the *Tools* menu in the Organizer. When the ValUI is started, a number of definition files are read, controlling the contents of the main window and some status windows. See "Definition Files" on page 2297 for more information.

No validator is started automatically by the ValUI in this way. The user must start a validator by selecting *Open* from the *File* menu, as stated in the text area of the main window, or by using the *Open* quick button.

A simple way to generate a validator, start the ValUI and open the validator is to click the *Validate* quick button in the Organizer.

When a simulator is started, a file selection dialog may be opened if the SDL system contains external synonyms. For more information, see "Supplying Values of External Synonyms" on page 2326 in chapter 54, *Validating a System*.

The Default Button Modules

The following tables list the default buttons in the button modules and the corresponding monitor command. See "Monitor Commands" on page 2232 for more information.

> **Note:**
>
> The buttons in the button modules are specified in the button definition file. If the default button file is not used, the button modules may be different than described here. See "Button and Menu Definitions" on page 2153 in chapter 50, *The SDL Simulator* for more information.

The *EXPLORE* Module

Button	Monitor command
Bit-State	Bit-State-Exploration
Exhaustive	Exhaustive-Exploration
Top	Top
Bottom	Bottom
Random Walk	Random-Walk
Tree Search	Tree-Search
Up	Up 1
Down	Down 1
Tree Walk	Tree-Walk
Verify MSC	Verify-MSC
Navigator	Show-Navigator
Break	Pressing `<Return>`

The *VIEW* Module

Button	Monitor command
Scope	Scope
Set Scope	Set-Scope
Scope Up	Scope-Up
Scope Down	Scope-Down
Call Stack	Stack
Ready Q	List-Ready-Queue
Process List	List-Process
Process	Examine-PId
Input Port	List-Input-Port
Signal	Examine-Signal-Instance

Button	Monitor command
Timer List	List-Timer
Timer	Examine-Timer-Instance
Variable	Examine-Variable

The *TEST VALUES* Module

Button	Monitor command
List Value	List-Test-Values
List Par	List-Parameter-Test-Values
List Signal	List-Signal-Definitions
Clear Signal	Clear-Signal-Definitions
Def Value	Define-Test-Value
Def Par	Define-Parameter-Test-Value
Def Signal	Define-Signal
Extract Sigs	Extract-Signal-Definitions-From-MSC
Clear Value	Clear-Test-Values
Clear Par	Clear-Parameter-Test-Values

The Menu Bar

This section describes the additional menus of the ValUI's menu bar in comparison with the SimUI. The menus common to both the ValUI and the SimUI is described in "The Menu Bar" on page 2138 in chapter 50, *The SDL Simulator*.

The menu bar contains the following menus:

- *File Menu*
  (See "File Menu" on page 2139 in chapter 50, *The SDL Simulator*.)
- *View Menu*
  (See "View Menu" on page 2139 in chapter 50, *The SDL Simulator*.)
- *Buttons Menu*
  (See "Buttons Menu" on page 2140 in chapter 50, *The SDL Simulator*.)
- *Log Menu*
  (See "Log Menu" on page 2141 in chapter 50, *The SDL Simulator*.)
- *Help Menu*
  (See "Help Menu" on page 15 in chapter 1, *User Interface and Basic Operations*.)
- *Commands Menu*
- *Options1 Menu*
- *Options2 Menu*
- Autolink1 Menu
- Autolink2 Menu

Additional Validator Menus

In addition to the standard SimUI menus, a few special validator menus are included in the menu bar. The menu choices in these menus simply execute a monitor command, i.e. they are functionally equivalent to buttons in the button modules. If the monitor command requires parameters, they are prompted for using dialogs in the same way as the command buttons.

The following tables list the default menu choices and the corresponding monitor command. See "Monitor Commands" on page 2232 for more information.

> **Note:**
>
> The additional menus in the ValUI are specified in the button definition file. If the default button file is not used, the additional menus may be different than described here. See "Button and Menu Definitions" on page 2153 in chapter 50, *The SDL Simulator* for more information.

Commands Menu

Menu choice	Monitor command
Toggle MSC Trace	MSC-Trace
Toggle SDL Trace	SDL-Trace
Show Report Viewer	Show-Report-Viewer
Show Coverage Viewer	Show-Coverage-Viewer
Show Navigator	Show-Navigator
Define Rule	Clear-Rule ; Define-Rule
Include Command Script	Include-File

Options1 Menu

Menu choice	Monitor command
Show Options	Show-Options
Reset	Reset
Default	Default-Options
Advanced	Define-Scheduling All ; Define-Priorities 1 1 1 1 1 ; Define-Max-Input-Port-Length 2 ; Define-Report-Log MaxQueuelength Off
State Space : Transition	Define-Transition
- : Scheduling	Define-Scheduling
- : Priorities	Define-Priorities
- : Input Port Length	Define-Max-Input-Port-Length
- : Transition Length	Define-Max-Transition-Length
- : Max Instance	Define-Max-Instance
- : Timer Progress	Define-Timer-Progress
- : Channel Queues	Define-Channel-Queue
- : Max State Size	Define-Max-State-Size
- : Symbol Time	Define-Symbol-Time
Report : Continue	Define-Report-Continue
- : Prune	Define-Report-Prune
- : Abort	Define-Report-Abort
- : Report Log	Define-Report-Log
MSC Trace Auto Popup	Define-MSC-Trace-Autopopup
Report Viewer Auto Popup	Define-Report-Viewer-Autopopup

Options2 Menu

Menu choice	Monitor command
Show Options	Show-Options
Bit-State : Hash Size	Define-Bit-State-Hash-Table-Size
- : Depth	Define-Bit-State-Depth
- : Iteration Step	Define-Bit-State-Iteration-Step
Random Walk : Repetitions	Define-Random-Walk-Repetitions
- : Depth	Define-Random-Walk-Depth
Exhaustive : Depth	Define-Exhaustive-Depth
Tree Search : Depth	Define-Tree-Search-Depth
MSC Ver : Timer Check Level	Define-Timer-Check-Level
- : Condition Check	Define-Condition-Check
- : Depth	Define-MSC-Verification-Depth
Autolink : Concurrent TTCN	Define-Concurrent-TTCN
- : Depth	Define-Autolink-Depth
- : Generation Mode	Define-Autolink-Generation-Mode
- : Global Timer	Define-Global-Timer
- : Hash Table Size	Define-Autolink-Hash-Table-Size
- : Options	Define-Autolink-State-Space-Options
- : Signal Mapping	Define-TTCN-Signal-Mapping
- : Test Cases Directory	Define-MSC-Test-Cases-Directory
- : Test Steps Directory	Define-MSC-Test-Steps-Directory
- : Test Steps Format	Define-TTCN-Test-Steps-Format

Autolink1 Menu

Menu choice	Monitor command
MSC : Save Test Case	Save-MSC-Test-Case

Menu choice	Monitor command
- : Save Reports	Save-Reports-as-MSC-Test-Cases
- : Save Error Reports	Save-Error-Reports-As-MSCs
- : Save Test Step	Save-MSC-Test-Step
- : List	List-MSC-Test-Cases-And-Test-Steps
- : Clear Test Case	Clear-MSC-Test-Case
- : Clear Test Step	Clear-MSC-Test-Step
Test case : Generate	Generate-Test-Case
- : Translate	Translate-MSC-Into-Test-Case
- : List	List-Generated-Test-Cases
- : Print	Print-Generated-Test-Case
- : Clear	Clear-Generated-Test-Case
- : Save	Save-Generated-Test-Case
- : Load	Load-Generated-Test-Cases
Test suite : Save	Save-Test-Suite

Autolink2 Menu

Menu choice	Monitor command
Configuration : Define	Define-Autolink-Configuration
- : Print	Print-Autolink-Configuration
- : Clear	Clear-Autolink-Configuration
- : Save	Save-Autolink-Configuration
- : Load	Include-File
Constraint : Define	Define-Constraint
- : List	List-Constraints
- : Rename	Rename-Constraint
- : Merge	Merge-Constraints
- : Parameterize	Parameterize-Constraint
- : Clear	Clear-Constraint
- : Save	Save-Constraint
- : Load	Load-Constraints
Timer : Define	Define-Timer-Declaration

The Command and Watch Windows

The Command window of the ValUI is identical to the SimUI (see "Command Window" on page 2147 in chapter 50, *The SDL Simulator*). The only difference is that the default commands to execute are "List-Process -" and "Print-Trace 1". The set of commands to execute are stored in a *command definition file* (see "Definition Files" on page 2297). The default command definition file can be changed with the Preference Manager.

The Watch window of the ValUI is identical to the SimUI (see "Watch Window" on page 2150 in chapter 50, *The SDL Simulator*).

The Navigator Tool

The Navigator is a separate tool in the ValUI that is used for navigating in the behavior tree. It can be opened in the following ways:

- By issuing the monitor command Show-Navigator
- By selecting *Show Navigator* from the *Commands* menu.
- By clicking the *Navigator* button in the button area.

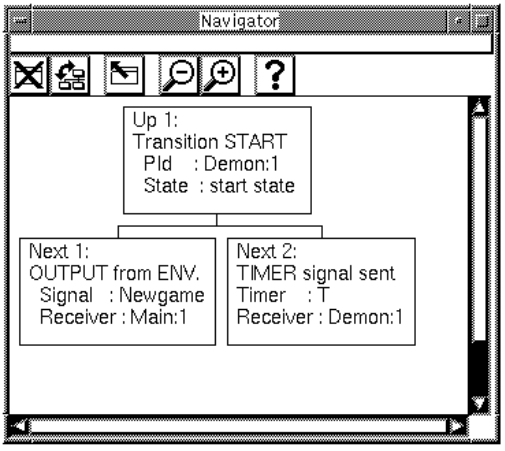

Figure 475: The Navigator window

The Navigator window consists of the tree area and the tool bar. There is no menu bar or status bar in the Navigator window.

Tree Area

The tree area of the Navigator window shows the structure of the behavior tree around the current system state. Each box, or node, represents a tree transition from one system state to another.

- The up node, always labelled *Up 1*, shows the tree transition leading to the current system state.
- The down nodes, labelled *Next 1*, *Next 2*, etc., show the possible tree transitions from the current system state. One of the down nodes may be labelled with three asterisks "***" to show which node is part of the current path, i.e., which part of the tree that already has been explored.

The text in the boxes contains the textual trace describing the tree transition. This may represent a complete or partial SDL process graph transition, depending on how the behavior tree is set up. The number of down nodes also depends on the structure of the behavior tree. This is determined by the state space options; see "State Space Options" on page 2396 in chapter 54, *Validating a System*.

Double-clicking a node in the Navigator executes the corresponding tree transition and moves one level up or down in the behavior tree. The current system state is changed and the Navigator window is updated to show the situation around the new system state.

Note:

Double-clicking a collapsed node (see below) **does not** expand the node; it always executes the corresponding transition.

The Navigator window is also updated whenever a monitor command is executed that changes the current system state.

Popup Menus

Each node in the node area has an associated popup menu.

Up 1	**On the up node:** Go up one level in the tree. This is the same as double-clicking the up node.
Up to top	**On the up node:** Go to the top of the behavior tree.

Continue Up	**On the up nodes**: Go up in the tree until more than one transition is possible (see Continue-Up-Until-Branch command).
Goto	**On the down nodes:** Go down this branch of the tree. This is the same as double-clicking the down node.
Continue	**On the down nodes:** Go down this branch of the tree until more than one transition is possible (see Continue-Until-Branch command).
Expand	Expand the collapsed node to show the down nodes. (Not applicable on a down node.)
Expand Substructure	The same as *Expand*.
Collapse	Collapse the node to hide the down nodes. A small triangle below the node shows that it is collapsed. (Not applicable on a down node.)

Quick Buttons

The following quick buttons are special to the Navigator tool. The general quick buttons are described in "General Quick-Buttons" on page 24 in chapter 1, *User Interface and Basic Operations*.

Close
Closes the Navigator tool.

Structure
Switches between a tree structure and a list structure in the node area.

Show Validator UI
Raises the Validator UI window.

The Report Viewer

The Report Viewer is a separate tool in the ValUI that is used for examining the reports generated during an automatic state space exploration. It is opened in one of the following ways:

- Automatically when an automatic state space exploration has finished, unless the option Report Viewer Autopopup is off
- By issuing the monitor command Show-Report-Viewer
- By selecting *Show Report Viewer* from the *Commands* menu.

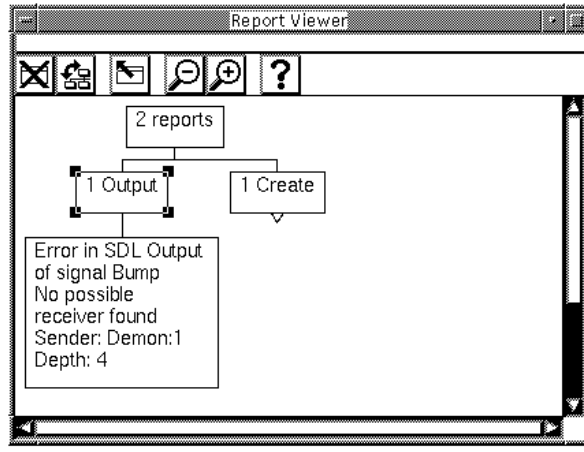

Figure 476: The Report Viewer

The Report Viewer window consists of the Report Area, containing Popup Menus, and the tool bar, containing Quick Buttons. There is no menu bar or status bar in the Report Viewer window.

Report Area

The report area of the Report Viewer shows the current reports from the latest state space exploration in the form of a report tree. The report tree contains three levels of nodes (see Figure 476):

- The top node shows how many reports there are. This node is always visible.

- The next level contains one node for each *report type*. The text shows the report type and how many reports of that type there are. When the Report Viewer is started, the report type nodes are shown, but they are collapsed (indicated by a small triangle below the node). For a list of possible report types, see "Rules Checked During Exploration" on page 2298.
- The bottom level contains one node for each actual report. The text shows the error or warning message, and the depth in the behavior tree where the report was generated. If the same report has been generated more than once, only one report is shown; the one generated at the lowest depth in the tree. When the Report Viewer is started, the report nodes are not visible.

Double-clicking a report node at the bottom level causes the Validator to "go to" the report, i.e., go to the system state where the report was generated. The following things happen:

- The textual trace from the transition where the report was generated is printed in the text area of the main window.
- If the Navigator tool is opened, it is updated to show the transition where the report was generated and the error message.
- An MSC Editor is opened, showing the MSC trace from the start state to the state where the report was generated. The MSC Editor is not opened if the option MSC Trace Autopopup is off.

Double-clicking the top node or a report type node will expand or collapse the node, depending on its state. A collapsed node hides its underlying nodes and is indicated by a small triangle below the node.

Popup Menus

Goto	**Only on report nodes:** Go to the report. This is the same as double-clicking the report node.
Expand	Expand the collapsed node to show the nodes in the next level of the tree. This is the same as double-clicking the collapsed node (top node or report type node). (Not applicable on a report node.)

Expand Substructure	Expand the collapsed node to show all nodes on all levels below it. (Not applicable on a report node.)
Collapse	Collapse the node to hide all nodes on all levels below it. This is the same as double-clicking the node (top node or report type node). (Not applicable on a report node.)

Quick Buttons

The following quick buttons are special to the Report Viewer. The general quick buttons are described in "General Quick-Buttons" on page 24 in chapter 1, *User Interface and Basic Operations*.

Close
Closes the Report Viewer.

Structure
Switches between a tree structure and a list structure in the report area.

Show Validator UI
Raises the Validator UI window.

Definition Files

In the ValUI, the syntax and contents of the definition files are the same as for the SimUI; see "Definition Files" on page 2152 in chapter 50, *The SDL Simulator*. The default file names for the definition files are `val_def.btns`, `val_def.cmds` and `val_def.vars`, respectively.

Rules Checked During Exploration

For each system state encountered during state space exploration, a number of rules are checked to detect errors or possible problems in the SDL system. If a rule is satisfied, a report is generated to the user.

The rules that are checked by the SDL Validator are the same ones as checked and reported by the SDL Simulator, with a few additions and differences.

Apart from the predefined rules, an additional rule can be defined by the user of the validator to check for other properties of the system states encountered. See "User-Defined Rules" on page 2308 for more information.

The rules are listed below, together with the reported error messages. The rules are grouped according to the corresponding report type, used in the Report Viewer in the Validator. The names of the report types listed below are the ones to be used in the monitor commands that define the report actions.

Deadlock

Report Type: Deadlock

```
Deadlock
```

All processes instances are waiting for some other process instance to act, implying that none of the processes will execute. This is referred to as a deadlock.

Implicit Signal Consumption

Report Type: ImplSigCons

```
Warning: Implicit signal consumption of signal
<signal>
```

A signal was sent to a process that was not able to handle (or save) the signal in the current state, so it was implicitly consumed.

Create Errors

Report Type: Create

```
Error in SDL Create: Processtype <type>
More static instances than maximum number of
instances.
```

There are more static instances defined than the maximum allowed number of instances.

```
Warning in SDL Create: Processtype <type>
Unsuccessful create. Number of instances has reached
maximum number.
```

An attempt has been made to create a new instance of a process type of which there is already the maximum number of allowed instances active. The maximum number is either the value defined by the command Define-Max-Instance, or the value defined in the SDL diagram, whichever is smallest.

Output and Remote Procedure Call Errors

Report Type: Output

```
Error in SDL Output of signal <signal>
No valid receiver found
```

An attempt was made to output a signal to an invalid PId expression.

```
Error in SDL Output of signal <signal>
No path to receiver
```

An attempt was made to output a signal to a PId expression. There exists, however, no path of channels and signal routes between the sender and the receiver that can convey the signal.

```
Error in SDL Output of signal <signal>
No possible receiver found
```

An attempt was made to output a signal without specifying a PId expression. When all paths or all paths mentioned in a via clause had been examined no possible receiver was found.

```
Error in SDL Output of signal <signal>
Several possible receivers found
```

An attempt was made to output a signal without specifying a PId expression. When all paths or all paths mentioned in a via clause had been examined several possible receivers were found.

```
Error in SDL Output of signal <signal>
Signal sent to stopped process instance
```

An attempt was made to output a signal to a PId expression that referred to a process instance which has performed a stop action.

```
Error in SDL Output of signal <signal>
Signal sent to NULL
```

An attempt was made to output a signal to a PId expression that was null.

```
Error in SDL Output of signal <signal>
Illegal signal type in output TRANSFER
```

An attempt was made to output a signal with the TRANSFER directive that was not the same signal as in the preceding input.

```
Error in remote procedure call: <name>
More than one process provides the remote procedure
```

An attempt was made to call a remote procedure in a system state where there are more than one possible process instance that can provide the remote procedure.

```
Error in remote procedure call: <name>
No process provides the remote procedure
```

An attempt was made to call a remote procedure in a system state where there is no possible process instance that can provide the remote procedure.

Max Queue Length Exceeded

Report Type: MaxQueuelength

```
Error in SDL Output of signal <signal>
Max input port length exceeded
```

The length of the input port of the receiving process has exceeded the value defined by the monitor command Define-Max-Input-Port-Length.

```
Error in SDL Output of signal <signal>
Max channel queue length exceeded
```

The length of the queue of the receiving channel has exceeded the value defined by the monitor command Define-Max-Input-Port-Length.

```
Error in channel output. Max input port length
exceeded
```

The length of the input port of the receiving process has exceeded the value defined by the monitor command Define-Max-Input-Port-Length.

```
Error in channel output. Max channel queue length
exceeded
```

The length of the queue of the receiving channel has exceeded the value defined by the monitor command Define-Max-Input-Port-Length

Channel Output Errors

Report Type: ChannelOutput

In addition to the following messages, information about the channel, signal, sender and receiver is also displayed.

```
Error in channel output. No valid receiver found
```

An attempt was made to output a signal to an invalid PId expression.

```
Error in channel output. No path to receiver
```

An attempt was made to output a signal to a PId expression. There exists, however, no path of channels and signal routes between the sender and the receiver that can convey the signal.

```
Error in channel output. No possible receiver found
```

An attempt was made to output a signal without specifying a PId expression. When all paths or all paths mentioned in a via clause had been examined no possible receiver was found.

```
Error in channel output. Several possible receivers
found
```

An attempt was made to output a signal without specifying a PId expression. When all paths or all paths mentioned in a via clause had been examined several possible receivers were found.

```
Error in channel output. Signal sent to stopped
process instance
```

An attempt was made to output a signal to a PId expression that referred to a process instance which has performed a stop action.

```
Error in channel output. Signal sent to NULL
```

An attempt was made to output a signal to a PId expression that was null.

Operator Errors

Report Type: Operator

The errors that can be found in operators defined in the predefined data types are listed below.

```
Error in SDL Operator: Modify! in sort Charstring,
Index out of bounds

Error in SDL Operator: Extract! in sort Charstring,
Index out of bounds

Error in SDL Operator: MkString in sort Charstring,
character NUL not allowed

Error in SDL Operator: First in sort Charstring,
Charstring length is zero

Error in SDL Operator: Last in sort Charstring,
Charstring length is zero

Error in SDL Operator: Substring in sort Charstring,
Charstring length is 0

Error in SDL Operator: Substring in sort Charstring,
Sublength is less than or equal to zero

Error in SDL Operator: Substring in sort Charstring,
Start is less than or equal to zero

Error in SDL Operator: Substring in sort Charstring,
Start + Substring length is greater than string
length.

Error in SDL Operator: Division in sort Integer,
Integer division by 0.

Error in SDL Operator: Division in sort Real, Real
division by 0.

Error in SDL Operator: Fix in sort Integer
Integer overflow
Second operand is 0

Error in SDL Operator: Mod in sort Integer
Second operand is 0

Error in SDL Operator: Rem in sort Integer
Second operand is 0
```

Range Errors

Report Type: Subrange

```
Error in assignment in sort <sort>:
<value> out of range
```

A variable of a restricted syntype is assigned a value out of its range.

Index Error

Report Type: Index

```
Error in SDL array index in sort <sort>:
<value> out of range
```

An array index is out of range.

Decision Error

Report Type: Decision

```
Error in SDL Decision: Value is <value>:
Fatal error. Analysis is not continued below this
node
```

The value of the expression in the SDL decision did not match any of the possibilities (answers).

Import Errors

Report Type: Import

Error during execution of an import statement. Supplementary information about remote variables and exporting processes is also given.

```
Error in SDL Import. Attempt to import from the
environment

Error in SDL Import. Attempt to import from NULL

Error in SDL Import. Attempt to import from stopped
process instance

Error in SDL Import. Several processes exporting
this variable

Error in SDL Import. The specified process does not
export this variable

Error in SDL Import. No process exports this
variable

Error in SDL Import. More than one process exports
this variable
```

View Errors

Report Type: View

Error during execution of view statement. Supplementary information about viewed variables and revealing processes is also given.

```
Error in SDL View. Attempt to view from the
environment

Error in SDL View. Attempt to view from NULL

Error in SDL View. Attempt to view from stopped
process instance

Error in SDL View. More than one process instance
reveals the variable

Error in SDL View. The specified process does not
reveal the variable.

Error in SDL View. No process instance reveals this
variable
```

Transition Length Error

Report Type: MaxTransLen

```
Max transition length exceeded
```

The maximum number of SDL symbols executed in one behavior tree transition is more than the value defined by the monitor command Define-Max-Transition-Length.

Non Progress Loop Error

Report Type: Loop

```
Loop Detected.
```

The Validator includes a mechanism for non-progress loop detection. A report will be generated if the Validator during state space exploration finds a loop in the state space which does not contain any progress transition. A progress transition is defined as a transition that either:

- contains communication with the environment
- contains a call to the function `xProgress()` in user-defined C code
- includes a timer expiration (see "Timer Progress" on page 2400 in chapter 54, *Validating a System*)
- includes a spontaneous transition (see "Spontaneous Transition Progress" on page 2401 in chapter 54, *Validating a System*).

Assertion Errors

Report Type: Assertion

```
Assertion is false: <user-defined string>
```

A user-defined action in the SDL system has called the function `xAssertError`. See "Using Assertions" on page 2387 in chapter 54, *Validating a System* for more information.

User Defined Rule

Report Type: UserRule

```
User-defined rule satisfied
```

A user-defined rule is evaluated to true.

Observer Errors

Report Type: Observer

```
Observer violation
```

A process defined as an observer process has not been able to execute a transition.

MSC Verification Errors

Report Type: MSCVerification
Report Type: MSCViolation

```
MSC <MSC name> verified
MSC sequence: <MSC name list>

MSC <MSC name> violated
Event: <SDL Event>
```

The MSC verification report is given when a state space exploration has reached a state where the execution trace from the root of the behavior tree to the current state satisfies the loaded MSC. In this case, "satisfies" means that:

- The execution trace must include all events that exist in the MSC.
- The execution trace must not contain any observable event that is not part of the MSC.
- The sequence of observable events in the execution trace must be consistent with the partial ordering of the events that is defined by the MSC.

An event on the execution trace is considered to be “observable” if it is either

- an input or output of a signal that is sent between process instances that correspond to different MSC instances, or
- a create of a process instance that corresponds to a different MSC instance than the creating process instance, or
- a set, reset or input of a timer, depending on the value set by the command Define-Timer-Check-Level.

Note:

Two process instances can correspond to the same MSC instance if the MSC is a system or block level MSC.

The MSC violation report is given during state space exploration for each generated execution trace that either:

- includes an observable event that is not part of the loaded MSC, or
- contains a sequence of observable events that is not consistent with the partial ordering of events defined by the MSC.

REF Errors

Report Type: RefError

```
No reference to dynamically allocated memory of
sort: <Sort Name>
```

The list of dynamically allocated data areas that is maintained by the Validator contains a data area that no SDL entity (like a variable, signal parameter etc.) references. This indicates a memory leak in the system.

```
No reference to dynamically allocated memory
```

The list of dynamically allocated data areas that is maintained by the Validator contains a data area that no SDL entity (like a variable, signal parameter etc.) references. This indicates a memory leak in the system. The lack of <Sort Name> in this report indicates that this is a newly allocated data area and there has been no previous system state with a reference to this data area.

```
Referenced data area not found:
Variable <Variable name> in process <Process name>
```

A REF variable (or signal parameter, structure field etc.) has been found that references a data area that is not allocated. To avoid this report always set the REF variables to NULL after the data area has been released.

```
Dereferencing a NULL pointer of sort: <Sort Name>.
```

An expression `XX*>` has been evaluated where `XX` was NULL.

```
Calling UserMalloc with parameter NULL
Calling UserFree with parameter NULL
```

The `UserMalloc` or `UserFree` functions has been called with NULL as a parameter which is illegal.

```
Calling UserFree without dynamically allocated
memory
```

`UserFree` has been called with a parameter that is not a pointer to one of the data areas in the list of the dynamically allocated data areas.

User-Defined Rules

User-defined rules are used during state space exploration to check for properties of the system states encountered. If a system state is found for which a user-defined rule is true, this will be listed among the other reports when giving the List-Reports command. During an exploration more that one user defined rule report can be generated. There will be one report for each value assignment that can be made to a rule. The value assignments are the values printed by the Print-Evaluated-Rule command.

A rule essentially gives the possibility to define predicates that describe properties of one particular system state. A rule consists of a predicate (as described below) followed by a semicolon (';'). In a rule, all identifiers and reserved words can be abbreviated as long as they are unique.

Note:

Only one rule can be used at any moment. If more than one rule is needed, reformulate the rules as one rule, using the boolean operators described below.

Predicates

The following types of predicates exist:

- Quantifiers over process instances and signals in input ports.
- Boolean operator predicates such as "and", "not" and "or".
- Relational operator predicates such as "=" and ">".

Parenthesis are allowed to group predicates.

Quantifiers

The quantifiers listed below are used to define rule variables denoting process instances or signals. The rule variables can be used in process or signal functions described later in this section.

```
exists <RULE VARIABLE> [: <PROCESS TYPE>]
[ | <PREDICATE>]
```

This predicate is true if there exists a process instance (of the specified type) for which the specified predicate is true. Both the process type and the predicate can be excluded. If the process type is excluded all process

instances are checked. If the predicate is excluded it is considered to be true.

```
all <RULE VARIABLE> [ : <PROCESS TYPE>]
[ | <PREDICATE>]
```

This predicate is true for all process instances (of the specified type) for which the specified predicate is true. Both the process type and the predicate can be excluded. If the process type is excluded all process instances are checked. If the predicate is excluded it is considered to be true.

```
siexists <RULE VARIABLE> [ : <SIGNAL TYPE>]
[ -  <PROCESS INSTANCE>] [ |  <PREDICATE>]
```

This predicate is true if there exists a signal (of the specified type) in the input port of the specified process for which the specified predicate is true. If no signal type is specified, all signals are considered. If no process instance is specified the input ports of all process instances are considered. If no predicate is specified it is considered to be true. The specified process can be either a rule variable that has previously been defined in an `exists` or `all` predicate, or a process instance identifier (`<PROCESS TYPE>:<INSTANCE NO>`).

```
siall <RULE VARIABLE> [ :  <SIGNAL TYPE>]
[ - <PROCESS INSTANCE>] [ | <PREDICATE>]
```

This predicate is true for all signals (of the specified type) in the input port of the specified process for which the specified predicate is true. If no signal type is specified all signals are considered. If no process is specified the input ports of all process instances are considered. If no predicate is specified it is considered to be true. The specified process can be either a rule variable that has previously been defined in an `exists` or `all` predicate, or a process instance identifier (`<PROCESS TYPE>:<INSTANCE NO>`).

Boolean Operator Predicates

The following boolean operators are included (with the conventional interpretation):

```
not <PREDICATE>
<PREDICATE> and <PREDICATE>
<PREDICATE> or <PREDICATE>
```

The operators are listed in priority order, but the priority can be changed by using parenthesis.

Relational Operator Predicates

The following relational operator predicates exist:

```
<EXPRESSION> = <EXPRESSION>
<EXPRESSION> != <EXPRESSION>
<EXPRESSION> < <EXPRESSION>
<EXPRESSION> > <EXPRESSION>
<EXPRESSION> <= <EXPRESSION>
<EXPRESSION> >= <EXPRESSION>
```

The interpretation of these predicates is conventional. The operators are only applicable to data types for which they are defined.

Expressions

The expressions that are possible to use in relational operator predicates are of the following categories:

- Process functions: Extract values from process instances.
- Signal functions: Extract values from signals.
- Global functions: Examine global aspects of the system state.
- SDL literals: Conventional SDL constant values.

Process Functions

Most of the process functions must have a process instance as a parameter. This process instance can be either a rule variable that has previously been defined in an `exists` or `all` predicate, a process instance identifier (`<PROCESS TYPE>:<INSTANCE NO>`), or a function that returns a process instance, e.g. `sender` or `from`.

```
state( <PROCESS INSTANCE> )
```

Returns the current SDL state of the process instance.

```
type( <PROCESS INSTANCE> )
```

Returns the type of the process instance.

```
iplen( <PROCESS INSTANCE> )
```

Returns the length of the input port queue of the process instance.

```
sender( <PROCESS INSTANCE> )
```

Returns the value of the imperative operator sender (a process instance) for the process instance.

```
parent( <PROCESS INSTANCE> )
```

Returns the value of the imperative operator parent (a process instance) for the process instance.

```
offspring( <PROCESS INSTANCE> )
```

Returns the value of the imperative operator offspring (a process instance) for the process instance.

```
self( <PROCESS INSTANCE> )
```

Returns the value of the imperative operator self (a process instance) for the process instance.

```
signal( <PROCESS INSTANCE> )
```

Returns the signal that is to be consumed if the process instance is in an SDL state. Otherwise, if the process instance is in the middle of an SDL process graph transition, it returns the signal that was consumed in the last input statement.

```
<PROCESS INSTANCE> -> <VARIABLE NAME>
```

Returns the value of the specified variable. If `<PROCESS INSTANCE>` is a previously defined rule variable, the `exists` or `all` predicate that defined the rule variable must also include a process type specification.

```
<RULE VARIABLE>
```

Returns the process instance value of `<RULE VARIABLE>`, which must be a rule variable bound to a process instance in an `exists` or `all` predicate.

Signal Functions

Most of the signal functions must have a signal as a parameter. This signal can be either a rule variable that has previously been defined in an `siexists` or `siall` predicate, or a function that returns a signal, e.g. `signal`.

```
sitype( <SIGNAL> )
```

Returns the type of the signal.

```
to( <SIGNAL> )
```

Returns the process instance value of the receiver of the signal.

```
from( <SIGNAL> )
```

Gives the process instance value of the sender of the signal.

`<RULE VARIABLE> -> <PARAMETER NUMBER>`

Returns the value of the specified signal parameter. The `siexists` or `siall` predicate that defined the rule variable must also include a signal type specification.

`<RULE VARIABLE>`

Returns the signal value of `<RULE VARIABLE>`, which must be a rule variable bound to a signal in a `siexists` or `siall` predicate.

Global Functions

`maxlen( )`

Gives the length of the longest input port queue in the system.

`instno( [<PROCESS TYPE>] )`

Returns the number of instances of type `<PROCESS TYPE>`. If `<PROCESS TYPE>` is excluded the total number of process instances is returned.

`depth( )`

Gives the depth of the current system state in the behavior tree/state space.

SDL Literals

`<STATE ID>`

The name of an SDL state.

`<PROCESS TYPE>`

The name of a process type.

`<PROCESS INSTANCE>`

A process instance identifier of the format `<PROCESS TYPE>:<INSTANCE NO>`, e.g. Initiator:1.

`<SIGNAL TYPE>`

The name of a signal type.

`null`

SDL null process instance value

```
env
```

Returns the value of the process instance in the environment that is the sender of all signals sent from the environment of the SDL system.

```
<INTEGER LITERAL>
true
false
<REAL LITERAL>
<CHARACTER LITERAL>
<CHARSTRING LITERAL>
```

Autolink Configuration Syntax

An Autolink configuration is created with the command Define-Autolink-Configuration. The syntax of an Autolink configuration is expressed below in EBNF format.

```
<Start>          ::= "Define-Autolink-Configuration"
                     <Configuration>
                     "End"

<Configuration> ::= { <TransRule> | <TSStructureRule> |
                      <ASPTypesRule> | <PDUTypesRule> |
                       <StripSignalsRule> <Function> }*

<TransRule>    ::= "TRANSLATE"
                   [ "SIGNAL" ] <AlternativeListOfTerms>
                   <TransRuleIf>* [ <TransRuleNoIf> ]
                   "END"

<TransRuleIf> ::= "IF" <Conditions> "THEN"
                   <TransRuleNoIf> "END"

<TransRuleNoIf> ::= { "CONSTRAINT"
                      <TransRuleConstraint> |
                      "TESTSUITE"
                      <TransRuleTestSuite> }*

<TransRuleConstraint> ::= { "NAME" <Term> |
                            "PARS" <ParameterList1> |
                            "MATCH" <ParameterList1> }*

<TransRuleTestSuite>  ::= { "CONSTS" <ParameterList1> |
                            "PARS" <ParameterList2> }*

<ParameterList1> ::= <Parameter1> { "," <Parameter1> }*

<Parameter1>     ::= "$" <Number> [ "=" <Term> ]

<ParameterList2> ::= <Parameter2> { "," <Parameter2> }*

<Parameter2>     ::= "$" <Number> [ "=" <Term> ]
                     [ "/" <Term> ]

<TSStructureRule> ::= "PLACE" <AlternativeListOfTerms>
                      <TSStructureRuleIf>*
                      [ <TSStructureRuleNoIf> ]
                      "END"

<TSStructureRuleIf> ::= "IF" <Conditions> "THEN"
                        <TSStructureRuleNoIf> "END"

<TSStructureRuleNoIf> ::= "IN" <Term> { "/" <Term> }*

<ASPTypesRule>      ::= "ASP-TYPES"
```

```
                            <SequentialListOfTerms> "END"
<PDUTypesRule>          ::= "PDU-TYPES"
                            <SequentialListOfTerms> "END"
<StripSignalsRule>      ::= "STRIP-SIGNALS"
                            <SequentialListOfTerms> "END"
<Function>              ::= "FUNCTION" <Identifier>
                            <Mappings> "END"

<Mappings>              ::= <Mapping> { "|" <Mapping> }*

<Mapping>               ::= <Conditions> ":" <Term>

<Term>                  ::= <Atom> { "+" <Atom> }*

<Atom>                  ::= "$" <Number> | "@" <Number> |
                            <Text> | <Identifier> |
                            <FunctionCall>

<FunctionCall>          ::= <Identifier>
                            "(" <SequentialListOfTerms> ")"

<SequentialListOfTerms>  ::= <Term> { "," <Term> }*

<AlternativeListOfTerms> ::= <Term> { "|" <Term> }*

<Conditions>            ::= <Condition> { "AND" <Condition> }*

<Condition>             ::= <Term> { "==" | "!=" } <Term> |
                            "TRUE"
```

For a detailed description of the semantics, see "Syntax and Semantics of the Autolink Configuration" on page 1433 in chapter 36, *TTCN Test Suite Generation*.

State Space Files

Using the command Save-State-Space, a Labelled Transition System (LTS) representing the state space generated during exhaustive exploration is saved to a file.

Syntax

The syntax of the generated LTS is (using BNF notation):

LTS	::=	'START:' StateId 'LTS:' TransList* 'STATES:' State*
TransList	::=	StateId ':' Event '-' [StateId] (',' Event '-' [StateId])* ';'
Event	::=	(Output | Input | Timeout | Internal)
Output	::=	'o(' Signal ',' '"' GraphRef '"' ')'
Input	::=	'i(' Signal ',' '"' GraphRef '"' ')'
Timeout	::=	't(' Timer ')'
Internal	::=	'x'
State	::=	'*****' StateId '*****' Process*
Process	::=	ProcessName ':' InstanceNo 'State:' StateName (VariableName ':' Value)* 'Input port:[' Signal (',' Signal)* ']' 'Timers:{' Timer (',' Timer)* '}' Procedure*
Procedure	::=	'Procedure' ProcedureName ':' 'State:' StateName (VariableName ':' Value)*
Signal	::=	SignalName '(' ParameterName (',' ParameterName)* ')'
Timer	::=	TimerName '(' ParameterName (',' ParameterName)* ')'

Lexical Elements

The lexical elements used above are:

StateId	An integer denoting a system state.
GraphRef	A string denoting a graphical reference to an SDL diagram.

ProcessName	An id denoting a process type.
InstanceNo	An integer denoting a process instance.
StateName	An id denoting a state in an SDL process graph.
VariableName	An id denoting a process variable.
Value	A string denoting a value for a variable of any defined SDL type.
ProcedureName	An id denoting a procedure.
ParameterName	An id denoting a signal or timer parameter.

Example 324: A Generated LTS

A simple example of a generated LTS:

```
START:121
LTS:
2456:o(sig1(true,3),"5 P1 1 80 100")-43567;
43567:i(sig2(1),"5 P1 1 80 120")-2467,
      i(sig2(2),"5 P1 1 100 120")-98567;
121:x-2456;
2467:x-27645;
98567:x-27645;
27645:i(sig2(1),"5 P1 1 80 120")-2467,
      i(sig2(2),"5 P1 1 100 120")-98567;
STATES:
***** 121 *****
P1:1 State:idle Parent:null Offspring:null
Sender:null i:0 Input port:[ ] Timers:{ }
***** 2456 *****
P1:1 State:idle Parent:null Offspring:null
Sender:null i:5 Input port:[ ] Timers:{ }
***** 43567 *****
P1:1 State:s1 Parent:null Offspring:null Sender:null
i:5 Input port:[ ] Timers:{ }
***** 2467 *****
P1:1 State:s1 Parent:null Offspring:null Sender:null
i:1 Input port:[ ] Timers:{ }
***** 98567 *****
P1:1 State:s1 Parent:null Offspring:null Sender:null
i:2 Input port:[ ] Timers:{ }
***** 27645 *****
P1:1 State:s1 Parent:null Offspring:null Sender:ENV
i:5 Input port:[ ] Timers:{ }
```

Restrictions

Restrictions on the SDL System

The restrictions on the SDL system that can be validated with the SDL Validator are basically the restrictions imposed by the SDL to C compilers. See "Restrictions" on page 2681 in chapter 57, *The Cadvanced/Cbasic SDL to C Compiler* for more information.

In addition to these general restrictions, the following restrictions are specific to the SDL Validator:

- Time is not represented in the state space explored by the validator. This makes the behavior of "Now" expressions special. In the SDL Validator, "Now" will always return 0.
- If user-defined types with dynamic memory allocation are used, the type must be implemented as a pointer to a data area. A function for freeing the memory must also be supplied. For more information, see "More about Abstract Data Types" on page 2634 in chapter 57, *The Cadvanced/Cbasic SDL to C Compiler*.

Restrictions on Monitor Input

There following restrictions apply to monitor input:

- A parameter to a monitor command may not contain more than 1,000 characters.
- **On UNIX**, control characters of different types may terminate the validation program. `<Ctrl+C>`, `<Ctrl+D>`, and `<Ctrl+Z>` are typical characters that might terminate a validation program.

Restrictions on Dynamic Checks

There are a number of dynamic checks that are not performed at all or performed at the C level by the C runtime system. A C runtime error will of course lead to the validation program being terminated. The following check is not made at the SDL level.

- Overflow of integer and real values are checked at the C level if the actual C system performs these checks. (These checks are likely not to be performed.)