MyCreativity Reference

For Southbeach Modeller 4

https://www.southbeachinc.com

support@southbeachinc.com

Beyond static diagrams, MyCreativity will transform your visualization and modelling experience. By adding simple rules and phrases, models become 'active'. As you click on, navigate or edit a model, the output generated guides problem-solving, innovation and many other creative tasks.

Anyone can use MyCreativity, from schoolchildren to PhD scientists. No programming skills required. Write readable rules that capture how you work and have Southbeach Modeller emulate hundreds of creative and problem-solving tools and methods.

useful+potential "What will you do with {this}?"

Table of Contents

1 Introduction

Southbeach Modeller is a unique visual modelling application. It includes an embedded rules engine called MyCreativity which generates ideas and reports from any visual model.

The rules engine provides three capabilities:

1. The ability to generate ideas (MyCreativity rulesets)
2. The ability to generate a report (MyReports templates)
3. The ability to generate combinatorial content (MyCues definitions)

Each builds on the other. The syntax for creativity rules is available within the body of reports. Similarly, rule patterns and macros are available when defining cues.

1.1 MyCreativity

What the generated sentences mean is up to the user. The model is a depiction of the system or situation. The generated sentences could be:

- Pointers to directions for problem-solving (ideation)
- A source of creativity (lateral thinking)
- Highlighting possible root causes (analysis)
- Content to share with colleagues or clients (reporting)
- Suggestions for further development of the model (elaboration)
- Specialized for a particular project or methodology (domain)

The rules engine is flexible, and it is simple for the user to write rules and rule sets for any purpose. No 'programming' skills are required. In addition, the software comes installed with a library of rules for those who do not wish to learn the rules language. These rulesets can also be used as templates and adapted by the user.

Rules can be as simple as this:

```
useful "Find a way to obtain more of {this}."
harmful "Find a way to obtain less of {this}."
```

An example of a slightly more complex rule could be:

```
produces(useful, harmful) "Find a way to prevent {from} from producing {to}."
```

And here is an example of a rule which matches contradictions in a system or situation and highlights how to think about them:

> produces(&a=useful, &b=useful) produces (&a, &c=harmful) "Find an alternative to {&a} that can provide {&b} but without producing {&c)."

A full description of the rules-language is given later in this document.

The user can define or select rules individually or in groups called rule sets. A rule set is, in effect, a way to define and apply a set of thinking to the model. For example, a consultant familiar with De Bono's 'Six Thinking Hats' may define a rule set like this:

```
#lateral thinking.6hats
* "What are the facts and figures around {this}?"
* "What is your gut reaction to {this}?"
* "What are the flaws and barriers in {this}?"
* "What are the benefits of {this}?"
* "Where does {this} take us? Are there creative alternatives?"
* "How should we think about {this}? Does this suggest a big picture?"
```

Rulesets are organized into a hierarchy and can be invoked individually or in combination. For example, taking the De Bono example, the user could turn on #lateral thinking.6hats.black and #lateral thinking.6hats.white as part of their analysis of a system or situation. The user can then also select between higher-level, more general, or lower-level, more detailed, and specific questions:

```
#lateral thinking.6hats.white
* "What do we already know about {this}? What are the facts?"
* "Who else knows about {this}? What light can they shed?"
* "Where is {this} used? What does that tell us?"
* "What is the life-cycle of {this}? Where did it come from? What does {this} turn into?"
* "How can we corroborate our hunches about {this}? What evidence is there?"
* "How can we quantify what we know about {this}?"
* "What disagreement is there about {this}? How can we be more objective?"
* "What more do we need to know about {this}?"
* "How can we get the new information about {this} that we need?"
```

The flexibility of the rules language, coupled with the richness of the Southbeach visual notation, means developing MyCreativity rulesets for virtually any analytical or problem-solving approach is possible. For example, here is a rule as part of a TRIZ contradiction matrix:

> increases(&a=*, @strength) decreases(&a, @power) "Try solving technical contradiction {&a} using the following TRIZ principles: 40 (Composite materials); 35 (Parameter changes); 3 (Local quality); 4 (Asymmetry); 10 (Preliminary action)

The rules language supports all features of the Southbeach Notation and user-defined or application-specific tags on each element of the model.

At run time, the user has control over how the rules fire when clicking around different parts of the model:

1. The ideation engine can be turned on or off. (Toolbar icon) This option can be convenient when the user does not wish to be distracted.

2. Each ruleset in the library can be individually turned on or off by the user.
3. The user has control of the 'extent' of pattern matching over agents and effects in the model, radiating out from the point at which they click.
4. Model-specific rules (i.e. rules embedded in the model itself) can be enabled or disabled.
5. The user can decide whether sentences generated in the application's creativity panel replace or are added to the current output. (There is also control of numbering, line spacing and quoted styles for the names of agents or effects inserted as macros into the sentences.)

The rules engine is smart and will only generate unique ideas or sentences. Only new ideas will be added to the creativity panel by the rules engine as the user explores different areas of the model.

The definition of 'extent' is as follows:

- 'Laser' only fires rules that match the 'clicked on' object or effect.
- 'Narrow' extends 'laser' by one connected object or effect.
- 'Extended' extends 'narrow'.
- 'Wide' extends 'extended'.
- 'Wider' extends 'wide'.
- 'Widest' matches all agents and effects it can find anywhere in the model by following connected links from the 'clicked on' object.

A note on performance:

Since visual models can be large and the user may have enabled many or all rulesets, the number of generated ideas can be significant. This potential performance issue will be especially true if the user has selected extent=widest.

The software is efficient and able to generate thousands of ideas almost instantly. However, in extreme cases, practical limits could be associated with the users' computers. In the Tools-Options menu, it is possible to configure the following:

- A creativity limit on the number of generated sentences per click (default 5000)
- Limitations on the generation of combinatorial creativity, 'cues'. More on 'Cues' later in this guide.

No particular IDE is required for users to edit or add rules and rulesets of their own. Instead, simple .txt files suffice. Likewise, notepad or Notepad++ will suffice. All installed libraries, and user-defined rulesets, can be found in the directory <documents>/southbeach/MyCreativity. Southbeach loads all rulesets found in that directory or its sub-directories at startup. Then, it merges the ruleset name hierarchy and presents the result to the user in the Toolbox - Creativity panel.

All installed library rulesets are, therefore, open to examination by the user. (For convenience, a right-mouse function at any point in the hierarchy will find and open the .txt file in which that ruleset exists.)

Another unique feature of Southbeach Modeller is the use of panel tabs. A model (a single .sbm file) can have as many ideation output panels as required. The engine adds newly generated ideas in whichever panel is visible at run time. Also, if the model consists of multiple sub-model (tabs), the software maintains an output panel for each sub-model and ideation panel combination. Again, whichever panels are visible at runtime determines where generated ideas are stored. If the append mode is on, no idea is lost until the user clears a specific tab. By contrast, if the replace mode is on, sentences are overwritten as the user clicks around the model. These different interactive modes serve complementary purposes:

1. To aggregate a set of unique ideas associated with all or a subset of model elements.
2. To interactively explore areas of the model, stimulated by ever-changing ideation.

Output panels are also the place where model-specific rules can be stored. Rulesets can be copied from the library, creating a new ideation panel. Or the user can add new panels as required, and ideation rules entered manually.

Output tabs divide into two: the upper part for editing rules and the lower part for the generated sentences. (Right-mouse in the panel, or F2)

1.2 MyReports

In addition to the interactive mode for generating creativity, the same rules language can be embedded in a report template and updated in a single click whenever a model associated with the report changes.

A report could consist mainly of user-supplied text and contain relatively few rules expanded into selected paragraphs. Or, the template could consist principally of rules surrounded by very little user-written text.

Reports can be held externally as .txt files or within Reports panels.

A report could be as simple as this:

```
[harmful]
```

This macro would generate a report comprising a list of the harmful elements of the model. Here is a slightly more complex example:

Actions we need to take include:

```
[useful "Find a way to increase {this}."]
[harmful "Find a way to decrease {this}."]
```

The full power of the rules language is available in the reporting language. In addition, several extensions, described in full in the reference guide later in this document, support:

- Numbering, lettering or bulleting generated sentences within the report output
- Reusing (i.e. calling) rulesets from the creativity library instead of writing out rules in full in the body of the report
- Formatting multi-line generated ideation.
- Inserting properties of the model, e.g. the model name, description and perspective, into the report template

Here is a simple report that uses these features:

> This is my report on ["{model.name}"] from the perspective of ["{model.perspective}"]:
>
> Background:
>
> ["{model.notes}"]
>
> The harmful elements include:
>
> [harmful]
>
> The goal is to solve the following problems:
>
> [produces(, harmful) “Find an alternative to {from}” ; problems]
>
> Ideas we should consider include:
>
> [increases(, harmful) "Find a way to prevent {from} increasing {to}" ; I]
>
> [decreases(, useful) "Find a way to prevent {from} decreasing {to}" ; continued]

The report will then include the notes associated with the model, a list of harmful elements, a numbered set of problem statements (P1, P2 etc.) and generated ideation (I1, I2 etc.).

Such a report could be held in an external .txt file and applied to any model. Reports can also be written and edited by the user in reporting panels. These work in the same way as ideation panels. The user can create as many reporting panels as needed. The panels divide in two as before: the upper part is the report definition, and the lower part is the generated output. A single click inside a panel (F5 or right mouse) refreshes the report output if the model has changed.

Reports run over complex models (all sub-models) or individual sub-models. Once again, whichever panels are visible when the user invokes a report determines which tab is updated. All templates and sub-model combinations are supported; no output is lost (unless the user clears a panel).

Because a report always processes a complete model (or sub-model), it can collect information as it goes and use this in following rules or macros later in the report. We call this MEMORY, and it works like a calculator's MI (memory in), MA (memory add) and MR (memory recall) keys. This simple yet powerful feature means that it is possible to write sophisticated reports that find and report on indirect effects between elements of a model where it would not be possible to write a matching pattern. The reference guide below describes the syntax used.

1.3 MyCues

Cues, a feature introduced in version 4 of Southbeach Modeller, automates the generations of combinations of creativity. Suppose, for example, that we needed to write a 6W1H set of questions. Rather than write each sentence out explicitly, we could write a cue definition:

```
-example
--6W1H
---question
----Who can
----Where should we
----When can we
----Why should we
----What should we focus on to
----Which area is relevant to
----How can we
---action
----analyze the problem?
----solve the problem?
```

The dashes indicate the grammar level. L1 organizes cues into a hierarchy. L2 represents the cue's name, in this example, 6W1H. L3 defines the parts of the sentence. This example has only two: the mode of the question and the action. L4 lists the combinations.

This cue definition generates 7x2 sentences.

```
* "Who can analyze the problem?"
* "Who can solve the problem?"
* "Where should we analyze the problem?"
* "Where should we solve the problem?"
* "When can we analyze the problem?"
* "When can we solve the problem?"
* "Why should we analyze the problem?"
* "Why should we solve the problem?"
* "What should we focus on to analyze the problem?"
* "What should we focus on to solve the problem?"
* "Which area is relevant to analyze the problem?"
* "Which area is relevant to solve the problem?"
* "How can we analyze the problem?"
* "How can we solve the problem?"
```

In this case, * has been added to the sentences. This syntax means the creativity rule will fire on any agent clicked on by the user. * means 'match anything' in the MyCreativity rules language.

The user can define cues with any number of grammar parts and L4 values per part. Millions of creative combinations are possible. Users can copy this to the operating system clipboard to paste it into another document or add the new rules to a creativity tab as model-specific creativity. They can also invoke cues interactively. The Cues engine selects random subsets as the user clicks around the model.

As with reports, the Cues engine builds on the power of the MyCreativity rules language. It is possible to define cues which:

1. Embed macros in the generated sentences.
2. Add patterns (to replace *) that determine whether a sentence 'fires'.
3. Select L4 combinations from the Southbeach tag groups and list resources.

Here, for example, is a cue definition which generates creative sentences for the SCAMPER lateral thinking method:

```
--SCAMPER
---Will
---@creativity:SCAMPER
---help solve {this}?
```

The @ syntax refers to a tag group, in this case, a set of SCAMPER keywords, provided as part of an extensive tag library delivered with the software. The syntax {this} refers to a macro that inserts the name of the selected agent into the sentence at runtime.

As with creativity rulesets, all cue sets are loaded at startup, defined in .txt files, and displayed to the user in the Toolbox panel, Cues tab. The user can enable or disable each cue set. And, based on the generated output, suppress or focus on subsets or individual sentences for the analysis at hand. The user can also disable the cues engine completely (toolbar icon).

In the Tool-Options menu, the user can set limits on 1) the number of sentences generated by the cues engine and 2) the random batch size returned as the user clicks around and explores a model.

That completes an overview of the creativity engine and the operation of the various panels and controls. The rest of this document is a reference guide explaining the full power of the MyCreativity engine and the syntax for rules, reports and cues.

2 Reference Guide

This reference guide describes how to write MyCreativity rules, rule sets, reports and cues. A key engine feature is the tight binding to the Southbeach Notation. For agents and effects, every keyword associated with the notation is available in the rules language. In addition, rules can refer to any tag or tag group annotating model elements. This unification means Southbeach Modeller is an out-of-the-box tool for numerous creative processes and a development platform for new applications.

2.1 The rules engine

Southbeach Modeller contains an embedded rules engine. The user writes or invokes library rules that generate creative suggestions from any visual model. These assist the user in analysis tasks such as design thinking, root cause analysis and innovation.

A rule is a pattern plus a sentence containing expandable macros. Keywords from the Southbeach notation are used to write patterns and macros. The engine supports every keyword and attribute in the visual notation. Patterns and macros can also match any user-supplied text in the model. User text is any agent name, effect name or label, tag or grid position on the canvas.

A rule set is a group of rules to support the analyst's intent. Rule sets can be named and organized into a hierarchy for easy access. Users turn on and off rule sets as they work through an analysis. As the user clicks around a model, rules are matched to the model content. Rules trigger within a defined extent the user sets at run time. Where a pattern matches the model within the scope, sentences are output. Macros embedded in the sentences are expanded as they are generated.

Creative suggestions (ideas) are listed in creativity panels within the application. Ideas generated are saved with the model in the tab in which they are generated. Model-specific rules can also be embedded in the creativity panels. The user can turn model-specific rules on or off at runtime.

In addition to this interactive mode, reports can also be generated. A report is a template containing embedded rules and reporting instructions. After any change to a model, the output can be re-generated and updated. Reports extend the pattern language to allow for the processing of indirect effects. Reports also provide formatting instructions for bulleted, numbered and lettered lists. Model properties, such as name, analyst notes and perspective, can be extracted into the report. Reports can be run over one or all models of a model file set.

Cues work in a similar fashion. The difference is that rules are generated on the fly as the user enables a particular set of cues.

2.2 The toolbox panel

Southbeach Modeller is supplied with an extensive toolbox (panel) of tags, grids, lists, creativity and cues. Each is displayed in a separate tab of the Toolbox. All such resources are defined in .txt files, e.g. mycreativity.txt. (Right mouse in a panel or tree to find and edit the associated file(s).)

Instructions at the top of each .txt file explain the syntax.

Use these files to create similar files and add them to the appropriate folder or sub-folder. All resources are loaded at startup, merged, and organized in the associated panel tab. If a syntax or processing error is detected, it is written to a log file (errors.log) in the same directory.

F5 (or right mouse) in a panel reloads the library files for that folder and its sub-folders.

Folder	**Contents**	**Panel/tab**
MyTags	Tags and tag groups	Toolbox - Tags
MyGrids	Grids/charts/swimlanes/pools/boxes	Toolbox - Grids
MyLists	Reusable resources	Toolbox - Lists
MyCreativity	Creativity rulesets	Toolbox - Creativity
MyCues	Cues definitions	Toolbox - Cues

2.3 MyCreativity

2.3.1 Naming rule sets

All rulesets are written as follows:

```
#name
pattern1 “output containing macros in {} or [] brackets”
pattern2 “output containing macros in {} or [] brackets”
etc.
```

Here is an example of a simple rule set:

```
#analyze.probe deeper
* "Consider {this}: What do you mean by this?"
* "Consider {this}: Tell me more about this."
* "Consider {this}: What else?"
* "Consider {this}: What other ways did you try so far?"
* "Consider {this}: What will you have to do to get the job done?"
* "Consider {this}: Is there something I should have asked that you need me to know?"
```

Notice how the # name defines a hierarchy of rule sets. These are displayed in a tree structure in the Creativity panel.

The * indicates that the rule should fire on any agent in the visual model. The {this} is a macro. It is replaced by the agent's name or effect, thereby completing the sentence.

2.3.2 Multi-line text

Multi line text for generated sentences is supported:

```
pattern "output text line 1\n
output line 2\n
output line 3"
```

2.3.3 Patterns supported

Five basic patterns are supported: agents, effects, chains, inward and outward. These are sufficient for the vast number of lateral thinking, root cause, design and problem-solving tools and methods supported by Southbeach Modeller.

Pattern	Explanation	Example
*	Match individual agents	
*(,)	Match effects	A effects B
(,*) *(*,*)	Match two effects with an object in common:	
(&a=,&b=*) *(&b=*,&c=*)	A ‘chain’ pattern where &b is the common object	A affects B which affects C
(&b=,&a=*) *(&c=*,&a=*)	An ‘inwards’ pattern where &a is the common object	B affects A and C also affects A
(&a=,&b=*) *(&a=*,&c=*)	An ‘outwards’ pattern where &a is the common object	A affects B and also affects C
[1] Please note that the engine cannot detect loops such as A effects B effects C effects A.		

Asterisk (*=any) can be replaced to match agents or effects with any attribute or combination of features supported in the Southbeach Notation.

Attributes can be combined (logical AND) using ‘+’. And negated (logical NOT) using an exclamation mark ‘!’.

Here is an example of each type of rule. In each case, the example sentence (creativity, ideas etc.) helps to understand the purpose of the pattern:

harmful “Find a way to remove {this} from the design.”

produces(useful, harmful) “How can we minimize the ability of {from} to increase {to}.”

produces(&a=*, &b=harmful) produces (&b, &c=useful) “Redesign the system to avoid the production of {&b} in the process of producing {&c} from {&a}”.

produces(&a=useful, &b=harmful) produces(&c=useful, &b) “Why do {&a} and {&b}

combine to produce harmful {&b}?”

produces(&a=useful, &b=useful) produces(&a, &c=harmful) “Find an alternative to {&a} that can produce {&b} without producing harmful {&c}.”

Notice the following keywords and macros in the examples:

- the keywords ‘useful’ and ‘harmful’
- the name of the effect ‘produces’ (an ‘increasing’ effect)
- {this} a macro to indicate the selected agent
- {from} and {to} to indicate the two agents involved in the effect ‘produces’
- {&a}, {&b} and {&c) names chosen for the agents involved so that they can be referred to in the pattern

In each table below, we introduce all the keywords in the Southbeach Notation and MyCreativity rules language. Then, at the end of each table, we present examples of rules using the keywords.

2.3.4 Keywords common to agents and effects

Keyword		
*		Match any agent or effect
useful		Match useful agents or effects
harmful		Match harmful agents or effects
neutral		Match neutral agents or effects
emphasis	Modifier	Match emphasized agents or effects
highlight	Modifier	Match highlighted agents or effects
insufficient	Modifier	Match insufficient agents or effects
sufficient	Modifier	Match sufficient agents or effects
potential	Modifier	Match potential agents or effects
dysfunctional	Modifier	Match dysfunctional agents or effects

Example rules:

emphasis(,goal) “{this} is emphasized because it effects a goal in the model.”

produces+dysfunctional(useful, useful+insufficient) “How can we modify the production of {to} by {from} to increase its reliability.”

!useful “{this} is not useful. Should it be?”

In the examples, ‘+’ combines keywords (AND). Any number of keywords can be AND’ed using ‘+’. An exclamation mark ‘!’ is used to negate keywords. The keyword must not be present for the rule to fire.

2.3.5 Combining and negating keywords

+	Two or more attributes must be present	useful+insufficient
!	Match if the attribute is not present	harmful+!risk

2.3.6 Keywords specific to agents

Some keywords only apply to agents. They are the types of objects and their modifiers.

Agent keyword		
agent	kind of agent	Match agents
issue	kind of agent	Match issues
choice	kind of agent	Match choices
action	kind of agent	Match actions
knowledge	kind of agent	Match knowledge
event	kind of agent	Match events
goal	modifier	Match goals
risk	modifier	Match risks
focus	modifier	Match agents with focus
historical	modifier	Match historical agents
surplus	modifier	Match surplus agents

Examples:

> produces(&a=choice, &b=action) produces(&a, &c=action) “Which action should we take: {&b} or {&c}?”
>
> historical+knowledge “Why do we no longer believe {this}?”

2.3.7 Matching text, tags and labels

The @ prefix allows for the definition of patterns that match ‘text’ or ‘tags’ in the visual model. This text includes the names of agents, tags and user-supplied effect labels. Therefore, the use of tags and effect labels opens the engine to defining rule sets for any domain.

Modeller has an extensive library of tags for all kinds of consulting and engineering purposes. The user can also define application-specific or domain-specific tag groups. Groups can be exclusive or inclusive, for example:

> Animals (exclusive): dog, cat, elephant, giraffe
>
> Traits (inclusive): fluffy, fierce, friendly, wild, domestic, small, large

MyCreativity provides three ways to match the text and tagged agents:

@<text>	Match any text	
@<effect label>	Match a user supplied effect label	
@<tag group>	Match all agents in the tag group	
@<tag>	Match agents with a tag (unqualified)	
@<tag group:tag>	Match agents with a tag (qualified)	
@<separation>	Match separated agents	See below for keywords
@<separation:value>	Match separated agents	See below for keywords
<separation>@<name>	Match separated agents	See below for keywords

(If a tag contains a space, it is still possible to match it. Replace the space character with an underscore, e.g. @to_do)

Examples:

> @cat “{this} is a cat.”
>
> !@dog “The dogs in the model other than {this} are {@dog}.”
>
> @dog+@fierce “{this} is a fierce dog that should not be domestic”
>
> increases(, @dog+@fierce) “Why does {from} increase the aggression in {to}?”
>
> role@CEO+harmful "Is {this} a problem for our CEO?"
>
> @degrades(, useful+@process) “What about step {from} is degrading process {to}?”
>
> produces(@engine, @power) “The {from} engine produces the form of power {to}.”
>
> @space:system “Should we move {this} to the supersystem?”
>
> increases(&a=*, &b=@triz:strength) decreases(&a, &c=@triz:power) “Consider replacing the system {&a} with a new system that does not require a trade-off between strength {&b} and power {&c}.”

Notes:

1. The use of fully-qualified tags. Defining a rule set for a specific methodology could be essential to differentiate generic forms of 'power' with a formalized definition of 'power', e.g. in TRIZ, the (Russian) Theory of Inventive Problem Solving.
2. Currently it is not possible to use * to negate a search for text or tags in a model.

2.3.8 Separation keywords

Separation keyword	**Example(s)**	**Application**
space	above, below, inside, outside	Spacial analysis
time	before, after, during past, present, future	Temporal analysis
parts	named parts	Structural analysis
perspective	viewpoint, opinion, argument	Perspective alignment
aspect	strength, colour, brand, quality	Qualitative analysis
role	employee, manager, CEO, CTO	e.g. Roles in a process
probability	inevitable, high, medium, low, remote	Probabilistic analysis
condition	below limit, above limit, nominal	Behavioral analysis
version	draft, final, v1, v2	Change management

Every Southbeach model can be overlayed onto a grid where the axes refer to separations. For example, a TRIZ 9boxes model would be defined thus:

rows (parts): supersystem, system, subsystem

columns (time): past, present, future

A SWOT chart could be defined thus:

boxes (aspects): strength, weakness, opportunity, threat

Since agents inherit tags from their position on grids and charts, rules can be written that respond to the movement of objects on the grid. For example:

parts@system "How is {this} affected by {@supersystem}?"

counteracts(@weakness, @opportunity) "How does weakness {from} limit opportunity {to}?"

@weakness "How can we convert weakness {this} into a strength?"

counteracts(@weakness, @opportunity) "How can we use our strengths: {@strength} to prevent weakness {from} limiting opportunity {to}?"

All of the effects in Southbeach Notation and their modifiers are supported in the rules language.

2.3.9 Keywords specific to effects

Effect keyword	Keyword type	Increasing/decreasing	Bi-directional?
produces	continuous	increases	
counteracts	continuous	decreases	
creates	discrete	increases	
destroys	discrete	decreases	
contributesto	participation	increases	
detractsfrom	participation	decreases	
consumes	resource	decreases	
stores	resource	increases	
becomes	transformation		
replaces	transformation		
causes	causes	increases	
prevents	causes	decreases	
implements	design		
specifies	design		
uses	resource	decreases	
related	relationship		bi-directional
is_a	inheritance/class		
opposed	contradiction		bi-directional
user_defined	user_defined		
NOT	negation		
necessary	modifier		
inevitable	modifier		
delay	modifier		
accelerate	modifier		
questionable	modifier		
excessive	modifier		
increases	derived		

decreases	derived		

The keywords ‘increases’ and ‘decreases’ are derived keywords. They match any effect that has, in a general sense, an increasing or decreasing effect in the model. This feature permits the writing of less fussy rules about the specific effects involved. For example:

> increases(,harmful) “Find a way to prevent {from} from increasing {to}.”

would match all three:

> produces(, harmful) “...”
> creates(,harmful) “...”
> contributesto(,harmful) “...”

where we might have written more specific rules such as:

> produces(, harmful) “Can we interrupt the process of {from} producing {to}?”
> creates(,harmful) “Why does {from} sometimes create defective widgets {to}?”
> contributesto(,harmful) “Can we isolate {from} from {to}?”

NOT requires special mention. NOT is not the same as ‘!’. NOT negates the effect. ‘!’ negates the pattern keyword. The following examples explain:

> NOT+produces(,) “{from} is not producing {to}.”
> NOT+produces(,) “Why is {from} not producing {to}?”
> etc.

as opposed to:

> !produces+harmful(,) “Whatever harmful is happening, it is not the production of {to} from {from}.”

2.3.10 Macro keywords

We have already seen macros such as {this}, {from} and {to} in the examples above. There are many more. The following tables list the macros available.

Syntax	**Explanation**	**Synonyms**
{} or []	Alternate brackets	
{*}	List of all agents	
{selected}	The clicked on ‘selected’ agent or effect	
{this}	The ‘matched’ agent or effect within the extent of pattern matching	
{from} {to}	Both ends of an effect	{arg1}, {arg2} {source}, {destination}

{&<argument name>}	The named argument in a pattern	
{<any agent pattern>}	List of the matching agents	
{<effect name>}	List of agents the selected agent is affecting	

Examples:

* "Are we missing anything from this list: {*}"

useful "Could {this} be impacted by any of these: {harmful+potential}?"

harmful "Does {this} have any other counteracting effects: {counteracts}?"

produces(,) "{this} is happening"

In the last example, the macro {this} refers to the effect because the pattern is an effect pattern. Thus, a summary of the effect is expanded in the sentence; in this case, 'A produces B is happening'. If more nuanced descriptions of effects and their modifiers are required, then the rule has to be written out more thoroughly. For example:

produces+insufficient(,) "Why is {to} being produced insufficiently by {from}?"

A set of keywords for each English tense of all effects is provided for convenience. See the later table. It is then possible to write:

produces+insufficient(,) "Why is {to} being {effected} insufficiently by {from}?"

2.3.11 When to use {selected} or {this}?

The difference between {selected} and {this} is often misunderstood, but the difference provides a powerful capability.

As the user clicks around the model, there is a {selected} agent. Since rules are applied within an extent (laser, narrow, extended, wide, wider, widest), matching occurs in the surrounding context. Those agents are {this}. This simple rule set illustrated:

```
#example.helpers
* "The selected object is {selected}"
* "Can {selected} be helped by {this}?"
```

Multiple sentences are output for the second rule as the user clicks around the model.

Always use {this} where you wish to refer to the agent where the rule matched, for example:

harmful+potential "Under what conditions could {this} be harmful?"

Use {selected} only in cases where you want to distinguish between the agent you 'clicked on' with the mouse and the agent against which the rule is matching. There will only be a difference between {this} and {selected} when you have set an extent greater than 'laser'. For example:

* “Have we understood how {selected} is affected by {this}?”

2.3.12 Macros for structure of the model (cause-effect links)

Macro	**Meaning**	**Context**	**Synonym(s)**
{inputs}	Agents that are inputs to the agent	matched agent	
{outputs}	Agents that are outputs the agent	matched agent	
{adjacent}	Agents that are connected to the agent by an effect (inputs, outputs and bi-directional relationships)	matched agent	{1hop}
{<n>hop} 1 to 5	Agents within <n> hops of the agent	matched agent	
{<name of an effect>}	Agents are related to the agent by that effect	linked agent	
orphan	An agent with no effects		{uninvolved}
leafnode	An agent with just one effect (in or out)		
start	An agent with only outgoing effects		{rootnode}
end	An agent with only incoming effects		{outcome}

Examples which use these macros:

* “Are any of these: {inputs} unnecessary in our design?”

* “Does {this} produce anything more than {outputs}?”

* “Should {this} have any other relationships other than {adjacent}?”

* “Things near {this} are {3hops}”

* “{this} is producing {produces}”

The ‘hops’ macro is useful when writing rules that explore the immediate vicinity of the selected ‘clicked on’ object. And also when analyzing large models or wishing to limit creativity generated when the extent is wide or widest. For example:

harmful(,) “How can we prevent {this} from causing harm in the following {3hops+useful}”

This rule will match any harmful effect in the model. The sentence generated will refer to useful agents (or effects) within three hops.

Effect names used as macros are helpful for many purposes. One of the most important is to refer to effects that are not directly involved in the pattern itself. For example:

harmful “In what additional ways other the production of {produces+harmful} is {this} harmful?”

2.3.13 Keywords extending problem-solving

The keywords below can be used either as pattern keywords or as macros. For example:

As a pattern:

problem+@serious “Solve {this}”

As a macro:

harmful “Is {this} contributing to any of the problems in our model: {problem}?”

Each of the keywords below also appears in the drop-down menu of the Highlight pattern tool. (Toolbar red flag icon.). The user can highlight the model according to each pattern and then follow the causes and effects across the model.

Keyword	**Pattern**	**Perspective**	**Synonym(s)**
orphan	An agent with no effects		{uninvolved}
leafnode	An agent with just one effect (in or out)		
start	An agent with only outgoing effects		{rootnode}
end	An agent with only incoming effects		{outcome}
unaffected	An agent unaffected by other agents		
noeffects	An agent with no effects on other agents		
solution	An agent with an outgoing useful effect	positive	
problem	An agent with an outgoing harmful effect	negative	
contradiction	An agent with both an outgoing useful effect and an outgoing harmful effect	solve	
conflicted	An agent with both an incoming useful effect and an incoming harmful effect	uncertainty	
unstable	An agent simultaneously being increased and decreased	uncertainty	
increasing	An agent increasing another agent	dynamics	
increased	An agent being increased by another agent	dynamics	
decreasing	An agent decreasing another agent	dynamics	
decreased	An agent being decreased by another agent	dynamics	
improving	An agent with useful outputs, e.g. increasing a useful agent or decreasing a harmful agent	situation	

improved	A useful agent being increased or a harmful agent being decreased	situation	
worsening	An agent with harmful outputs, e.g. decreasing a useful agent or increasing a harmful agent	situation	
worsened	A useful agent being decreased or a harmful agent being increased	situation	
pro	A useful agent being increased	positive	{benefit}
con	A harmful agent being increased	negative	{drawback}
improvingfactor	A useful agent that is decreasing a harmful agent	positive	
worseningfactor	A harmful agent that is increasing a harmful agent	negative	
enabler	A useful agent that is increasing a useful agent	positive	
barrier	A harmful agent that is decreasing a useful agent	negative	
silverlining	A useful agent (the silver) being increased by a harmful agent	positive	
necessaryevil	A harmful agent (the evil) increasing a useful agent or decreasing a harmful agent	positive	
problemsolved	A ‘problem’ agent being decreased	positive	
compromisedsolution	A ‘solution’ agent being decreased	negative	

Advantages of derived keywords:

1) Derived keywords allow for more complex patterns:

compromisedsolution “Why do we need {this} in the design?”
produces(, contradiction) “{this} is a cause of a contradiction. How can we remove {this}?”
produces(solution, problem) “Why does solution {from} cause problem {to}?”
produces(, outcome+harmful) “How can we change {from} to avoid harmful outcomes?”

2) Derived keywords can also make writing reports easier. A report can be as simple as:

My model contains the following problems:
[problem ; P]
and solutions:
[solution ; S]
The following contradictions must be addressed:
[contradiction ; C]

Southbeach is a very flexible tool. Not all users develop models in the style or methodology. The definition of derived keywords (i.e. pre-defined patterns) will not always be intuitive in the context of an unusual model. Nevertheless, the fact that the additional keywords exist may help in writing creativity rules and reports that would otherwise not be possible.

2.3.14 The {random} macro

{random}	Select 1 from the matching objects	

The macro 'random' is used by itself or in combination with other keywords. For example:

harmful "Can {random} help {this} become useful?"

When the user clicks on a harmful element in the model, a sentence asks whether a randomly selected element could help? Click again, and a new sentence is generated. If we then change the rule to select only useful agents, it would be:

harmful "Can {random+useful} help {this} become useful?"

The following rule is powerful when extent is 'widest':

* "Can {useful+random} be used to solve {harmful+random}?"

The rule fires on any agent selected. Depending on how the dice fall, a different number of sentences will be generated on each click. A randomly chosen useful element is paired with a randomly selected harmful agent, a powerful form of creativity from a single rule.

2.3.15 Macros to generate the correct English form of effects

Using these macros in creativity sentences makes it possible to write rules that generate the correct form of English text for all of the different effects in Southbeach Notation.

Macro	**English form**	**Example for 'produces' effect**
{effect}	Verb use	Does A produce B?
{effects}	Verb use	A produces B
{effecting}	Process	A is producing B
{effected}	Past tense	In the past, A produced B
{effectedby}	Reversed	B is produced by A
{effector}	Noun (actor)	A is the producer of B
{effection}	The act of	The production of B by A

While it is possible to write:

produces(,) “{from} produces {to}”

it is also possible to write:

produces(,) “{from} {effects} {to}”
produces(,) “How does {from} {effect} {to}?”
etc.

The following table lists the form of English returned by the macros for each of the effects in the Southbeach Notation:

Macro	**{effect}**	**{effects}**	**{effecting}**	**{effected}**	**{effectedby}**	**{effector}**	**{effection}**
	verb	*verb*	*process*	*in the past*	*reversed*	*noun (actor)*	*the act of*
produces	produce	produces	producing	produced	is produced by	producer	production
counteracts	counteract	counteracts	counteracting	counteracted	is counteracted by	counter	counteraction
creates	create	creates	creating	created	is created by	creator	creation
destroys	destroy	destroys	destroying	destroyed	is destroyed by	destroyer	destruction
contributesto	contribute	contributes to	contributing to	contributed to	receives from	contributor	contribution
detractsfrom	detract from	detracts from	detracting from	detracted from	is detracted by	detractor	detraction
consumes	consume	consumes	consuming	consumed	is consumed by	consumer	consumption
stores	store	stores	storing	stored	is stored by	store	storage
becomes	become	becomes	becoming	became	is derived from	past form	transformation
replaces	replace	replaces	replacing	replaced	is replaced by	replacement	substitution
causes	cause	causes	causing	caused	is caused by	cause	causation
prevents	prevent	prevents	preventing	prevented	is prevented by	preventer	prevention
implements	implement	implements	implementing	implemented	is implemented by	implementer	implementation
specifies	specify	specifies	specifying	specified	is specified by	design	specification
uses	use	uses	using	used	is used by	user	use
related	relate to	is related to	related to	related to	is related to	relative	relation
is a	inherit	is a	being a	was a	describes	example	description
opposed	oppose	opposes	opposing	opposed	opposes	opposite	opposition
user_defined	affect	affects	affecting	affected	is affected by	affector	effect [1]

[1] To remember when to use ‘affect’ and ‘effect’: A is for action (affect); E is for the end result (effect).

[2] Some of these macros do not work well if the effect is negated (i.e. not occurring). In those cases, we recommend writing the rule explicitly for NOT (negated) effects. For example:

NOT+produces(,) “{to} is NOT produced by {from}”

2.3.16 A note on writing style for creativity rules

Southbeach Modeller does not impose a specific methodology on users. The analyst may adopt a formal style they are familiar with, or they may play it fast and loose with how they name agents and effects..

Consider this sentence:

> The table supports the cup

or this:

> The engine produces power

or this:

> The power is generated by the engine

There are different ways to model these in Southbeach Notation. What are the agents? Nouns or verbs? What are the effects? Relationships or processes? What part of the text is the agent? What part is the effect? Or is the whole sentence a single agent?

How an analyst models a situation or system can determine how they should write rules that generate creative ideas. A complete discussion of this is beyond the scope of this guide. However, it has been shown that however the model is notated, it can be helpful to write the rules with an article in front of macros such as {this}, {from} and {to}. The following examples illustrate:

> harmful "Find a way to reduce (the) {this}"
>
> counteracts(, harmful) "How can (the) {from} be modified so that it decreases more of (the) {to}?"

Notice how the article 'the' has been placed in brackets (). This tip helps a reader scan the generated sentences in one of two ways, which is more forgiving of whether the macros expand to nouns, verbs, relationships, processes or entire phrases. The English cannot be 100% 'correct' in all instances because the engine knows anything about the names the analyst has given to the agents and the effects in the model.

So which of these is best?

1. produces(, harmful) "Limit (the) {from} in order to limit (the) {to}"
2. produces(, harmful) "Limit the {from} in order to limit the {to}"
3. produces(, harmful) "Limit {from} in order to limit {to}"

There is no correct answer. Users will adopt their modelling style and a suitable format for their creativity rules. The library rules distributed with the software vary in style.

2.4 MyReports

MyReports builds on the capability of the MyCreativity rules engine.

A report consists of template text with embedded macros. Macros are expanded line by line or as comma-separated lists in the body of the report. The generated sentences (ideas, issues etc.) can be styled (e.g. bulleted lists, numbering). Macros can be as simple as a single notation keyword or as complex as any examples in this guide.

Southbeach Modeller provides 'Reports' tabs for saving reports with the visual model to which they apply or in external .txt files to use the report with any model.

Syntax	**Meaning**	**Note**
//	Comment line, no report output	
[]	Expand macro line by line	
{}	Expand macro comma separated	
<pattern>	Expand into report	
<pattern> "<sentence>"	Expand into report	Sentence can embed macros
model.name	Expand into report	
model.perspective	Expand into report	
model.notes	Expand into report	
#<rule set>	Expand rule set into report	Specify the # name of the rule set from the MyC tree
; <numbering style>	See below	

Here is a short report that includes both forms of brackets, macros inside sentences, numbering output lines and calling on a library rule set:

> My report on [{model.name}] from perspective [{model.perspective}]
>
> Our goals are: {goal}. The risks are {risk}. We must perform these actions:
>
> [increases(, harmful) "Find a way to prevent {from} increasing {to}" ; actions]
>
> [decreases(, useful) "Find a way to prevent {from} decreasing {to}" ; actions]
>
> Let's try some green hat thinking:
>
> [#lateral thinking.6hats.green ; ideas]
>
> Background:

[{model.notes}]

The size and complexity of the report generated will depend on the number of agents and effects in the visual model and how they match the patterns specified in the rules.

Reports vary hugely. Often, a report will consist mainly of text written by the consultant and will only call upon the engine to insert generated ideas in an appendix. The appendix section could be as simple as:

Appendix – our ideas

[#TRIZ.solve.contradictions]

Other times, the report will consist of detailed calls to the engine with virtually no surrounding text.

2.4.1 Memory function for indirect effects

As a report is generated, it is possible to 'collect' the results of rules and use them in subsequent rules further down the report. Here is a simple example:

The factors limiting our goals are:

[decreases(MI, goal) "{from}"]

The factors that are increasing those are:

[increases(, MR) "{from}"]

The keyword 'MI' 'collects' the matching objects in the rule. These can then be referred to in a subsequent pattern using the keyword 'MR'.

Keyword	Definition	Example
MI	Collect (memory in)	
MR	Recall (memory out)	
MA	Accumulate (add to collected set)	

The three keywords can be used in conjunction with any other MyCreativity keywords. For example:

[counteracts(useful+MI, useful) "{from} is countering {to}"]

What can we do to remove any of these functions:

[MR]

Using MI, MR and MA, it, therefore, becomes possible to follow indirect effects in the model, further side-effects, secondary causes and relationships, and so on, forever. For example:

[increases(MI, risk) “{from} is a primary cause of risk in the system”]

[increases(MI, MR) “{from} is a secondary cause of risk in the system”]

[increases(,MR) “{from} is a tertiary cause of risk in the system”]

Another way to lay out that report could be:

The primary causes of risk in the system are:

[increases(MI, risk) “{from}”]

The secondary causes of risk in the system are:

[increases(MI, MR) “{from}”]

The tertiary causes of risk in the system are:

[increases(MI, MR) “{from}”]

Note that negation of the recalled set of objects is also supported. For example:

[counteracts(, !MR) “{from} is countering {to}”]

Where {to} refers to the objects other than the objects returned from the previous MI or MA. Thus it is also possible to refer to objects that do not match patterns. For example:

[useful+@activity+MI]

!MI would then to refer to any object that is not a useful activity.

2.4.2 Numbering, lettering and bulleting

As well as generating numeric and alpha (A-Z) numbered lists, MyReports provides keywords for numbering styles. This can help encourage consistent reports. The syntax is as follows:

[<pattern> ; <style>]

[<pattern> “<sentence containing macros>” ; <style>]

Note that in Southbeach v3, a colon : was used. In v4, this has been changed to a semi-colon. Use of colon will now generate an error if used to specify numbering style. Colon is now reserved for specifying fully-qualified tags, e.g. @taggroup:tag.

Keyword	**Style**		**Keyword**	**Style**	
A thru Z	e.g. F	F1, F2, F3 …			
dashed	-		harmfuls	H	
ddashed	--		ideas	I	

bullets	*		issues	I	
dbullets	**		opportunities	O	
chevrons	>		options	O	
dchevrons	>>		proposals	P	
numbered		1 2 3…<n>	problems	P	
alpha		A, B, C…Z, A2, B2, C2 …	questions	Q	
assertions	A	A1, A2, A3…A<n>	strengths	S	
actions	A	ditto	threats	T	
barriers	B	B1, B2, B3…B<n>	tasks	T	
choices	C	etc.	usefuls	U	
constraints	C		weaknesses	W	
decisions	D		pros	+	+1, +2, +3…
goals	G		cons	-	-1, -2, -3…

Control of numbering and output:

zero	zeros the numbering counters
continued	continue the previous numbering style
blank	run rule/macro but suppress the output

The keyword ‘blank’ is useful in conjunction with MI, MA and MR. For example:

[increases(MI, risk) “this should never appear” ; blank]

[increases(MI, MR) “this should never appear” ; blank]

[increases(,MR) “{from} is a tertiary cause of risk in the system” ; numbered]

2.5 MyCues Reference

MyCues is a powerful new feature of Southbeach Modeller, introduced in v4. The new cues engine processes combinatorial creativity from simple grammar definitions. This feature can generate almost infinite creativity and complements the curation of bespoke MyCreativity rulesets.

Whereas MyCreativity rule sets can be given #names, cue sets are defined using a simple hierarchy reflecting the structure of a 'grammar'. Sentences have 'parts' and 'parts' have alternate values. This structure allows all cues to be presented to the user in the Cues panel according to the grammar.

The values from which combinations are generated are explicitly specified or taken from the names of tags or items in a list. Thus, all library resources are available to the cues engine.

Syntax	Explanation	
//	Comment line, no output	
-	Cue area (for organizing cue sets)	L1 in hierarchy
--	Cue set name	L2
---	Cue part (the sentence grammar)	L3
----	Combination words/phrases	L4
@	expand a tag_area:tag_group	The tag values obviate the need to specify the L4 combinations
@@	expand a list	The members of the list obviate the need to specify the L4 combinations
!	Suppress an area, cue set, part or value	During merging of Cue files
---^<pattern>	<creativity pattern>	Any pattern supported by the engine
----^<pattern>	<creativity pattern>	Any pattern supported by the engine
\	Suppresses leading spaces	Allows the next part to butt up against the previous part

Here is an example of a simple cue called 'investigate':

```
--investigate
---How should we
---action
----investigate
```

```
----collect evidence for
----build a case around
----secure a prosecution
----prevent in the future
---a
---@crime:crime
---crime?
```

It generates creativity rules of the form:

> * "How should we <action> a <crime> crime?"

The actions are listed in the cue definition. The crimes are taken from the tag library. In the example, the sentence does not contain any macros, nor are any patterns used to determine which sentences are generated as the user selects objects in the model. By default, * has been added so the sentence will 'fire' on any selected object. To improve this example, we can add a pattern and macros:

```
--investigate
---^harmful(@evidence,)
---How should we
---action
----investigate
----collect evidence for
----build a case around
----secure a prosecution
----prevent in the future
---(the) {this}?
---Is there evidence that that (the) {from} is the cause of the
---@crime:crime
---crime?
```

The generated sentences are:

> harmful(@evidence,) "How should we <action> (the) {this}? Is there evidence that (the) {from} is the cause of the <crime> crime?"

When the user applies this cue to the visual model, interactively or in the body of a report, the rules 'fire' only when an agent tagged 'evidence' is the source of a harmful effect. Moreover, the macros {this} and {from} are substituted into the sentence at run time. (As with all rulesets, the cues are applied within the extent the user sets from the selected object (laser, narrow, wider etc.). A typical output might be:

How should we build a case around (the) hacking produces altered electronic records? Is there evidence that (the) hacking is the cause of the white collar crime?

3 Installed Content

Southbeach Modeller comes pre-installed with 100s of creativity rulesets. The source of these scripts can be found here (.txt files).

<documents>\southbeach\MyCreativity

<documents>\southbeach\MyReports

<documents>\southbeach\MyCues

Three examples are given below:

3.1 MyCreativity example

```
#lateral thinking.6hats.black
* "Will {this} work for us?"
*(,) "Will {this} work for us?"
* "Why can't we do {this} now? What should we in preparation?"
*(,) "Why can't we do {this} now? What should we in preparation?"
* "What is wrong with {this}? What might cause this to break?"
*(,) "What is wrong with {this}? What might cause this to break?"
* "What are the risks around {this}? Is this safe?"
*(,) "What are the risks around {this}? Is this safe?"
* "What is {this} in conflict with? Think about ideals, standards and  conventions"
*(,) "What is {this} in conflict with? Think about ideals, standards and  conventions"
* "How will {this} impact on other activities?"
*(,) "How will {this} impact on other activities?"
* "What distractions does {this} create?"
*(,) "What distractions does {this} create?"
* "Who will prevent, stop or derail {this}?"
*(,) "Who will prevent, stop or derail {this}?"
```

3.2 MyReports example

This is my report on ["{model.name}"] from perspective ["{model.perspective}"]

Our goals are {goal}. The risks are {risk}. Choices we need to make include:

[choice ; C]

Each raises issues:

[produces(choice, issue) “{this}” ; I]

There are several harmful effects in this situation which we need to mitigate:

[harmful(,) “{this}” ; H]

The main areas of improvement lie in:

[increases(*, harmful) "Finding a way to prevent {from} increasing harmful {to}." ; I]

[decreases(*, useful) "Finding a way to prevent {from} decreasing harmful {to}." ; I]

Actions we have already decided are:

[counteracts(action+!potential, harmful) "Intervention {from} will help to mitigate {to}"; A]

Potential new actions are:

[action+potential "Can intervention {this} help us?"; P]

It would be helpful for our next session to use Six Thinking Hats, focused on the harmful effects in the situation. We believe that a combination of black hat thinking and yellow hat thinking would be appropriate.

For the priority areas, questions we could ask include:

Judgement questions:

[harmful(,@priority) #lateral thinking.6hats.black: judgement ; Q]

Benefits questions:

[harmful(,@priority) #lateral thinking.6hats.yellow: benefits ; Q]

Finally, we have noted the following about the situation:

["{model.notes}"]

3.3 MyCues example

// e.g. Who can develop further {this} from the perspective of security against the metric testability?

```
--6W1H
---question
----who can
----where should we
----when can we
----why should we
----what should we
----which area of
----how can we
---intent
----improve
----challenge
----elaborate
----explain
```

----describe
----understand more about
----simplify
----analyse
----optimise
----extend
----reimagine
----create a plan for
----create
----create a better
----obviate the need for
----protect
----simulate the next generation of
----test
----revamp
----probe deeper into
----develop further
----play with
----think about
----review
----action
---[this] from the perspective of
---perspective
----process
----organisation
----location
----data
----application
----technology
----enterprise
----service
----functions
----financials
----usability
----operations
----teaming
----delivery
----cost
----profit
----policy
----customer need
----partners
----suppliers
----security
----systems
----solutions
---against the metric

---metric
----effectiveness
----efficiency
----value for money
----sustainability
----success / goals
----quality
----availability
----predictability
----consistency
----integration
----servicability
----manageability
----accessability
----appeal
----functions
----features
----performance
----reliability
----security
----testability
----usability
----competiveness
----economic value
----vulnerability
---\?